Science of Computer Programming 248 (2026) 103381

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Metric information mining with metric attention to boost software defect prediction performance

Yongchang Ding [a], Wei Han [a], Zhiqiang Li [b], Haowen Chen [c], Linjun Chen [a], Rong Peng [a,*], Xiao-Yuan Jing [a,d,*]

[a] School of Computer Science, Wuhan University, Wuhan, Hubei, China
[b] School of Computer Science, Shanxi Normal University, Xi'an, China
[c] School of Cyber Science and Engineering, Information Engineering University, Zhengzhou, China
[d] Guangdong Provincial Key Laboratory of Petrochemical Equipment Fault Diagnosis, School of Computer, Guangdong University of Petrochemical Technology, Maoming, Guangdong, China

ARTICLE INFO

Keywords:
Software defects prediction
Feature selection
Feature transformation
Attention mechanism
Metric data mining
XAI

ABSTRACT

In the field of software engineering, defect prediction has always been a popular research direction. Currently, the research on traditional software defect prediction mainly focuses on metric features, which are derived from various descriptive rules. Many researchers have proposed a large number of defect prediction models based on these metric features and various framework models. However, the problem of data scarcity has severely hindered the development of the field. Therefore, this work proposes a new method, namely the Metric Attention Module (MAM), which excavates the correlations within the metric data features, between features, within modules, and between modules. By learning new data representations, MAM guides the model's learning process and ultimately improves the model's performance without changing the network framework structure. Additionally, the method is interpretable.

In this work, experiments were conducted in various task environments and on different datasets, all resulting in varying degrees of improvement. In the context of within-project defect prediction (WPDP), experiments with the MAM data model showed an average improvement of 14.7% in Accuracy, 15.9% in F1 score, 23.7% in AUC, and 65.1% in MCC. In cross-project defect prediction (CPDP), under more complex task environments, the model demonstrated excellent performance across multiple standard datasets. Compared to the baseline models and training results, the F1, Accuracy, and MCC scores improved by approximately 40%, 20%, and 50%, respectively.

1. Introduction

In the field of software engineering, defect prediction is a critical task that involves identifying potentially faulty modules early in the development process [1][2]. Researchers aim to identify as many potential defective program modules as possible within limited technical conditions, enabling sufficient testing resources to be allocated accordingly.

Current software defect prediction primarily relies on historical development data and machine learning techniques to forecast defect-prone modules. Prediction approaches are generally categorized into three types: within-project defect prediction (WPDP),

* Corresponding authors at: School of Computer Science, Wuhan University, Wuhan, Hubei, China.
E-mail addresses: rongpeng@whu.edu.cn (R. Peng), jingxy_2000@126.com (X.-Y. Jing).

https://doi.org/10.1016/j.scico.2025.103381
Received 27 February 2025; Received in revised form 19 June 2025; Accepted 6 August 2025

Available online 13 August 2025

cross-project defect prediction (CPDP), and heterogeneous defect prediction (HDP). Among these, metric-based prediction methods are the most widely used. They typically involve three stages: extracting metric data, training predictive models, and generating predictions based on the model output.

Recent years have seen the development of diverse metric features tailored to different systems, such as code metrics [16], process metrics [17], and historical defect metrics [18]. These advancements have significantly enhanced the application of machine learning in defect prediction. However, the performance of these metrics is often influenced by dataset variations in practical tasks, sometimes leading to conflicting results. For example, Fenton [3] and Gyimothy [4] have expressed differing opinions on the value of size and complexity metrics; the former argues that these metrics fail to provide meaningful insights, while the latter contends they offer strong predictive performance.

These discrepancies highlight a critical challenge in metric-based defect prediction. Over the years, a large number of metric features have been proposed and extracted based on standardized software measurement criteria. However, in practice, many of these features may be redundant, noisy, or task-irrelevant due to mechanical or uniform feature transformation pipelines. This becomes especially problematic in cross-project (CPDP) and heterogeneous (HDP) settings, where the overlap or compatibility between metric sets across projects is often very low. As a result, models trained on one project often perform poorly when applied to others, limiting generalization.

Rather than continuously designing new metric features or relying on manual selection, a more promising direction is to enhance model learning by mining latent information from existing metric data. By leveraging internal feature correlations, models can gain deeper insight from the data without requiring handcrafted feature engineering.

To address similar data challenges, domains like medical and financial modeling often use data augmentation (DA) to improve generalization [22,19,20]. While DA in image and text data typically involves semantic-preserving transformations [23,21], metric data in defect prediction lacks such adaptable augmentation techniques. Inspired by this, we propose a method that preserves original metric representations while mining internal correlations using a self-attention mechanism [5]. This allows the model to assign differentiated weights and extract deeper semantic information from existing features.

1.1. Motivation

As software systems become increasingly complex, the prediction and detection of software defects are growing in importance. Software defects not only affect the quality and reliability of software but can also lead to significant economic losses and safety issues. Consequently, effective prediction and detection of software defects have emerged as critical challenges in the field of software engineering.

However, several issues remain unresolved.

Existing models often rely on various metric features for defect prediction, but the dependence on single data types means that model performance is heavily influenced by the quality of these metric features. This reliance on single data types has become a key factor limiting progress in the field. For instance, Yang et al. utilized historical defect data for defect prediction and found that the model's performance significantly declined in projects with insufficient historical data [6].

In addition, existing defect prediction models often lack interpretability. Although some machine learning and deep learning models demonstrate strong predictive performance, their complex internal mechanisms make it challenging to explain how decisions are made. This poses a challenge for developers and managers, who cannot understand the basis of the model's predictions and thus struggle to trust or validate the results. For example, Jian Li et al. used deep neural networks for defect prediction, achieving higher prediction accuracy but facing challenges due to the "black box" nature of the model, which made it difficult for developers to identify specific influencing factors [7]. Matloob et al. proposed an ensemble learning method that integrates multiple models for defect prediction. While the performance improved significantly, the complexity of the ensemble approach made it hard to interpret the model's decision-making process [8].

1.2. Contribution

In this study, we propose a module called the Metric Attention Module (MAM). This module enhances model performance by uncovering the complex relationships between metric features through the exploration of both local and global information.

Capturing complex metric relationships MAM is capable of capturing the complex relationships between local and global features, including intra-project feature relationships, inter-project feature relationships, correlations within metric features, and correlations between distinct metric features. By modeling these comprehensive relationships, MAM acquires richer data to guide model training. In multiple model experiments under multi-task environments such as WPDP and CPDP, MAM significantly improved test performance.

Utilizing weights to measure feature importance and enhance method interpretability MAM leverages the weights from the information mining process to analyze the importance of features, thus enhancing the model's interpretability. By analyzing the attention weights, it is possible to clearly understand which features played a key role in the prediction. This is crucial for understanding the model's decision-making process and for further improving the model.

A novel data mining approach as a supplementary data augmentation method in software defect prediction MAM enhances the basic data through its unique relationship extraction approach, improving data utilization efficiency. This method provides a new data augmentation technique for the field of software defect prediction. The definition and establishment of new relationships help in better understanding and utilizing data, alleviating issues related to data scarcity.

The remainder of this paper is organized as follows. In Section 1, we introduce related work on software defect prediction in Subsection 1.1 and explain the main contributions of this study in Subsection 1.2. In Section 2, we provide a detailed description of MAM and the definitions of various relationships. Section 3 elaborates on the dataset, evaluation metrics, model, and experimental design. Sections 4 and 5 present the research results and discuss the main insights.

1.3. Related works

Software defect prediction has long been a focus of research in software engineering. With the rise of machine learning, metric-based approaches have become the predominant strategy. These approaches rely on software metrics—quantitative descriptions of code or development processes—to train models for identifying defect-prone modules. Existing research can be broadly categorized into three areas: metric design, metric selection, and metric transformation.

1.3.1. Software metrics

Early studies on defect prediction introduced traditional code-based metrics, such as Halstead [9] and McCabe complexity metrics [10], to quantify procedural code characteristics. As object-oriented programming gained popularity, new metrics were developed to capture inheritance, cohesion, and coupling. Representative examples include C&K metrics [11] and MOOD metrics [12].

In parallel, process-oriented metrics were proposed to capture information beyond source code, incorporating aspects such as developer activities, historical bug records, and code changes. For instance, Bhattacharya et al. [13] used social network analysis to model developer interactions, while Lee et al. [14] introduced Micro Interaction Metrics (MIMS) based on developer behavior. Jiang et al. [15] further proposed personalized metrics (PCC) to tailor predictions to individual developers. Although process metrics can improve localization accuracy, they often require extensive manual data collection and are difficult to scale.

Despite the diversity and effectiveness of many proposed metrics, practical challenges remain. Metric extraction often follows standardized but rigid procedures, which may yield noisy or irrelevant features for specific tasks. This issue becomes more pronounced in cross-project (CPDP) and heterogeneous (HDP) defect prediction, where low overlap and poor alignment between metric sets limit model generalizability.

1.3.2. Metric selection

To address redundancy and irrelevance in metric data, many studies have explored feature selection techniques. These aim to identify the most informative subset of metrics from the original set, thereby improving both performance and interpretability.

Menzies et al. [24] employed data-driven selection strategies to reduce feature dimensionality and improve model robustness. The CLAMI method [25] integrates clustering-based analysis to jointly select metrics and modules. In the context of heterogeneous defect prediction, Yu et al. [26] proposed a distribution-matching approach: metrics are selected and aligned between source and target projects based on statistical similarity. These methods demonstrate that careful metric selection can help mitigate feature mismatches across projects.

1.3.3. Metric transformation

Another line of work focuses on transforming existing metric features into new representations to address heterogeneity and improve model adaptability. Unlike selection, transformation methods alter the feature space, often at the cost of losing interpretability.

Classical transformations include Principal Component Analysis (PCA) [27], Linear Discriminant Analysis (LDA) [28], and Multidimensional Scaling (MDS) [29], which reduce dimensionality while preserving structural properties. Menzies et al. also applied logarithmic transformations and normalization techniques to mitigate skewness in code metric distributions.

For CPDP tasks, Nam et al. [41] proposed Transfer Component Analysis (TCA) to align feature spaces between projects. Jing et al. [42] extended this by introducing CCA+, which builds a unified metric representation through canonical correlation alignment. Li et al. [43] further incorporated cost-sensitivity and kernel mapping in CTKCCA to handle data imbalance, and later proposed EMKCA to manage non-linearity and class distribution differences. These studies confirm that metric transformation plays a vital role in addressing distribution shifts across datasets.

1.3.4. Limitations and our contribution

Although extensive research has been conducted on metric selection and transformation, several challenges persist. Feature selection methods often rely on heuristics or dataset-specific assumptions, while transformation techniques can compromise model interpretability and add computational complexity. Moreover, both approaches typically operate on individual feature vectors without explicitly modeling relationships among features.

In contrast, our work explores a new direction: mining the intrinsic relationships within metric data through a self-attention mechanism. By learning weighted dependencies between features, our proposed Metric Attention Module (MAM) enhances prediction performance and interpretability without introducing additional data or altering the original metric space. This approach provides a lightweight yet effective alternative to traditional feature engineering methods.

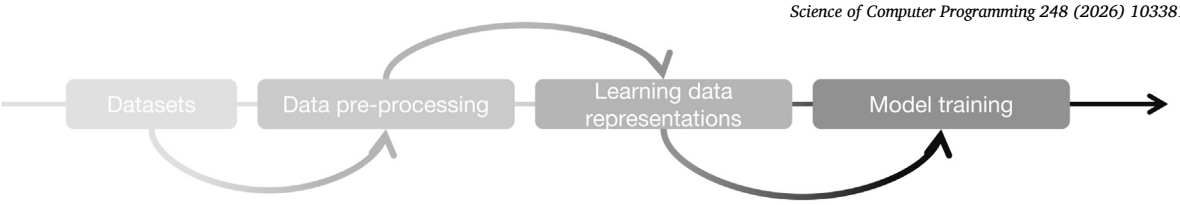


Fig. 1. Overall flow chart.

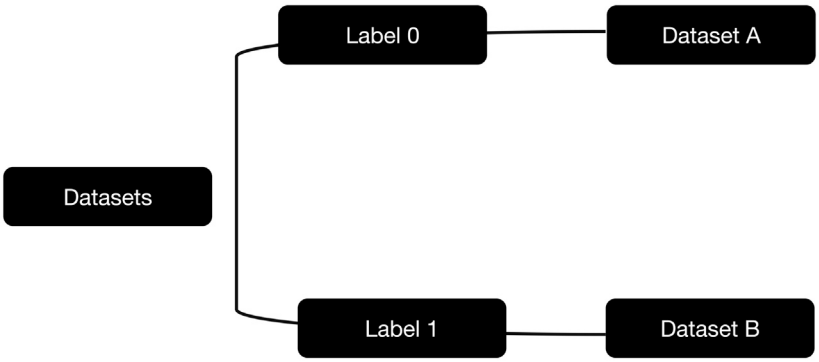


Fig. 2. Data pre-processing-Training data.

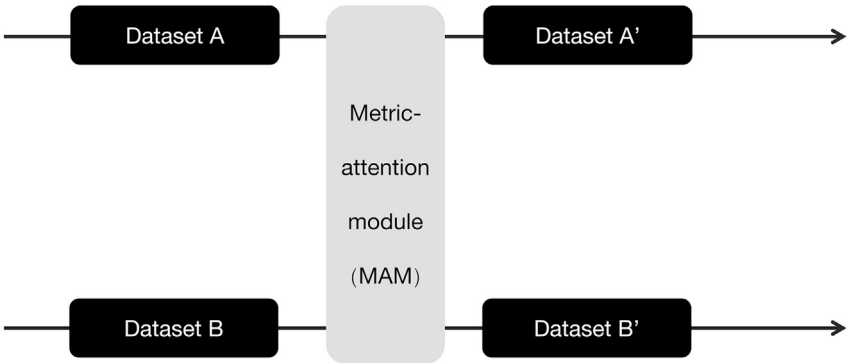


Fig. 3. Learning data representations.

2. Methodology

2.1. Technology pathway general process

As shown in Fig. 1, the complete technical workflow for this study is divided into four stages: Datasets, Data Pre-processing, Learning Data Representations, and Model Training (see Figs. 2–4). Through the data processing across these four stages, the final model exhibits a significant performance improvement compared to the baseline model.

2.2. Data pre-processing

During the data preprocessing stage (see Fig. 2), the data is first divided based on different labels to enable separate processing. This approach aims to enhance the model's ability to independently handle and learn distinct data representations during the learning process.

$$Dataset\,A = (x_i, y_i) \text{ where } y_i = 0, \quad i \in [1, n] \tag{1}$$

$$Dataset\,B = (x_i, y_i) \text{ where } y_i = 1, \quad i \in [1, n] \tag{2}$$

Suppose there is a training dataset Datasets containing n samples. Each sample consists of a feature vector and a corresponding label, where the label indicates whether a defect exists: 0 represents no defect, and 1 represents a defect.

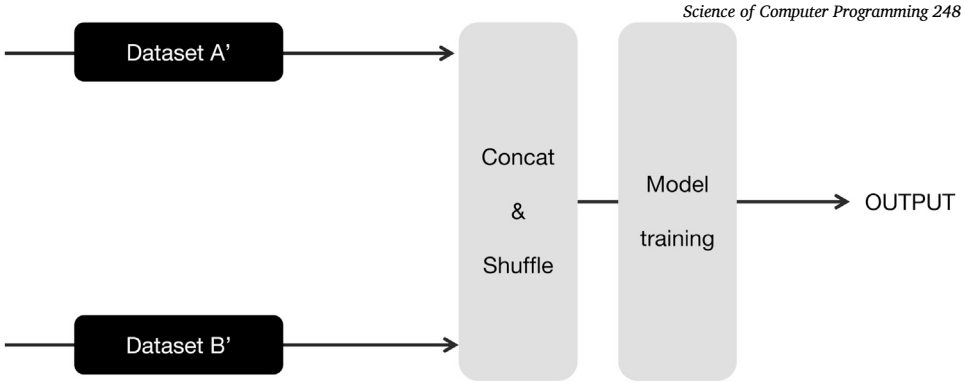


Fig. 4. Model training.

2.3. Learning data representations

In the previous stage, the experimental data was divided into two subsets, each corresponding to a different label. In the second stage, known as "Learning Data Representationssee" (see Fig. 3), the authors employed a Metric-Attention Module (MAM) to independently learn the representations of each subset. This process resulted in the generation of new data representations that capture the distinct characteristics of the data associated with each label.

$$Dataset\,A' = \int (Dataset\,A), \quad Dataset\,B' = \int (Dataset\,B) \tag{3}$$

$Dataset\,A$ and $Dataset\,B$ represent subsets containing samples with different labels, respectively. Let $\int$ denote the mapping function of the MAM module, and $Dataset\,A'$ and $Dataset\,B'$ represent the representations learned through the MAM module.

Here, $\int$ represents the mapping function of the MAM module, and $Dataset\,A'$ and $Dataset\,B'$ are the learned representations after processing.

2.4. Model training

After the second stage of data representation learning, the third stage focuses on model training (see Fig. 4). First, the data processed in the second stage is combined (via concatenation and shuffling) to form a new dataset. This dataset incorporates the newly learned data representations from the MAM module. By feeding this enhanced dataset into the model, it is possible to extract richer and more precise information, thereby improving the overall performance of the model.

$$Dataset\_New = Shuffle(Concat(Dataset\,A', Dataset\,B')) \tag{4}$$

$$OUTPUT = Model(Datasets\_New) \tag{5}$$

Here, $concat(\cdot)$ denotes the concatenation operation, $shuffle(\cdot)$ denotes the shuffling operation, and $model(\cdot)$ denotes the model training process.

2.5. Metric-attention module

In this work, the authors propose a general module called the "Metric-Attention Module," which aims to improve the efficiency of data utilization and enable more comprehensive learning of data representations. The authors introduce a Self-attention mechanism, continuing the attention mechanism's ability to mine both local and global information, in order to better extract metric feature information.

At the same time, based on a naive understanding of metric selection and metric transformation methods, the author hopes that under the influence of this module, the shortcomings or imperfections of both approaches can be eliminated, while merging their advantages. Therefore, to improve interpretability, the author introduces the Self-attention mechanism, extending the idea of attention mechanisms to solve the problem of insufficient interpretability in metric data processing.

The Self-attention Mechanism is a crucial concept in the field of deep learning, especially in natural language processing. The advantages of self-attention primarily lie in its ability to capture long-range dependencies, facilitate parallel processing, provide richer sequence representations, and offer flexibility in data processing.

These features are exactly what this research aims to leverage. In this study, the author utilizes the attention mechanism to easily mine relational information between features. The work, based on metric feature data, introduces the concepts of "Metrics within the instance", "Within a metric feature", "Inter-metric features", and "Inter-instance features" to describe the role that a single metric feature value plays within modules, metric intervals, and instances (see Algorithm 1 for pseudo-code).

Algorithm 1: Metric Attention Module (MAM).

Input: $X \in \mathbb{R}^{m \times n}$: metric data with m instances and n features
$Y \in \{0,1\}^m$: binary defect labels
$\alpha, \beta, \gamma, \delta$: attention weights for Corr1–Corr4 (sum to 1)
Output: $Z \in \mathbb{R}^{m \times n}$: enhanced data representation

Step 1: Split dataset based on labels: $X_0 \leftarrow \{x_i \mid y_i = 0\},\ X_1 \leftarrow \{x_i \mid y_i = 1\}$;
foreach $X_i \in \{X_0, X_1\}$ **do**
    Project features into query, key, value spaces:;
        $Q_i = X_i W_Q, \quad K_i = X_i W_K, \quad V_i = X_i W_V$;
    Compute attention-based correlation matrices:;
        Corr1: intra-instance feature attention ;
        Corr2: inter-instance similarity ;
        Corr3: intra-metric temporal correlations ;
        Corr4: inter-metric cross-feature correlation ;
    Fuse multiple attention relations:;
    $A_i = \alpha \cdot \text{Corr1} + \beta \cdot \text{Corr2} + \gamma \cdot \text{Corr3} + \delta \cdot \text{Corr4}$;
    Apply attention weights to feature values:;
    $Z_i = A_i \odot X_i$; /* Hadamard product */
Concatenate and shuffle: $Z = \text{Shuffle}(\text{Concat}(Z_0, Z_1))$;
return Z ;

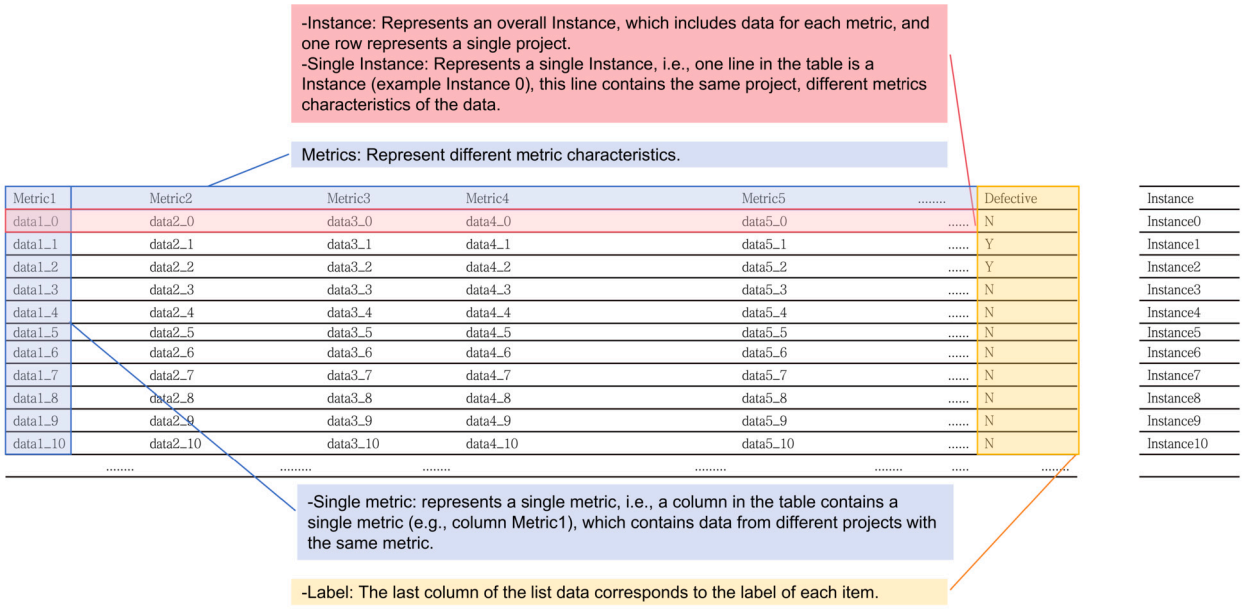


Fig. 5. Structure of Software Defect Prediction Standard Data Set. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

2.5.1. *Relationship definitions*

First, the author provides a unified definition for the standard dataset specific to software defect prediction, as shown in table in Fig. 5. The first row (blue area) represents the names of different metric data, while the last column (orange area) serves as the label. Each row (red area) represents a specific project (see Fig. 5). Based on this structure, the author defines the following terms tailored to the special data structure and the characteristics of software defect prediction tasks:

- **Instance**: Represents an instance, which includes various metric data. Each row corresponds to a single instance.
- **Single instance**: Refers to a single instance, where each row in the table represents an instance (e.g., Instance 0). This row contains data for the same instance but with different metric features.
- **Metrics**: Represents different metric features.
- **Single metric**: Refers to a single metric feature, where each column in the table represents a metric feature (e.g., Metric1). This column contains data for the same metric feature across different projects.
- **Label**: The last column in the dataset corresponds to the label for each project.

The relationships are categorized into the following types: “Inter-metric feature relationships within Instances”, “Inter-instance feature relationships”, “Correlation within metric features”, and “Correlation between single metric features”.

Table 1
Grouped interpretation of Q, K, and V in the two attention types.

Type	Scope	Input Domain	$Q/K/V$ Construction and Meaning
Feature-oriented Attention			
Corr1	Feature ↔ Feature (within instance)	Single instance (row of X)	Q, K, V are projected from each feature vector: $Q = X_i W_Q$, $K = X_i W_K$, $V = X_i W_V$. Captures intra-instance feature dependencies.
Corr3	Instance ↔ Instance (same metric)	Single metric (column of X)	For metric j: $Q = X_{\cdot j} W_Q$, etc. Models global patterns of metric across instances.
Instance-oriented Attention			
Corr2	Instance ↔ Instance	Entire matrix X	$Q = \text{Mean}(X_i)W_Q$, $K = \text{Mean}(X_j)W_K$, $V = \text{Mean}(X_j)W_V$. Measures instance similarity globally.
Corr4	Metric ↔ Metric	Entire matrix X	$Q = \text{Mean}(X_{\cdot j})W_Q$, etc. Captures inter-feature patterns aggregated across instances.

- **Corr1:Inter-metric feature relationships within instance**: This refers specifically to the relationships between different metric data within a single instance (e.g., data1_0, data2_0). For example, in a practical instance, the relationship between "lines of code" and "code complexity".
- **Corr2:Inter-instance feature relationships**: This refers to the relationships between a single instance and another instance (e.g., Instance1 and Instance2). In a practical instance, this could represent the similarity of code complexity between different instances.
- **Corr3:Correlation within metric features**: This specifically refers to the correlation of feature data under a single metric (e.g., Metric1) within an instance (e.g., data1_0, data1_1). For example, in a project, this could be the correlation of lines of code across different instances.
- **Corr4:Correlation between single metric features**: This refers to the correlation between a single metric feature and another metric feature (e.g., Metric1 and Metric2). For example, across multiple instances, this could represent the correlation between lines of code and code complexity.

2.5.2. Basic operational logic of relationship mining

The relational information extraction in the Metric-Attention Module (MAM) is grounded in the self-attention mechanism, which computes the relevance between entities through learned query, key, and value representations. All four defined correlation types (Corr1–Corr4) operate under a unified computational framework, differing only in the construction of their input spaces.

To formalize this process, let $X \in \mathbb{R}^{m \times n}$ be the input metric feature matrix, where m is the number of instances and n is the number of features. Each row corresponds to an instance and each column to a metric feature. The attention operation can be expressed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V \tag{6}$$

Here, Q, K, and V are matrices derived via linear projections of the input data: $Q = XW_Q$, $K = XW_K$, and $V = XW_V$, where W_Q, W_K, and W_V are learnable weight matrices, and d is the dimensionality of the projected space. This operation allows the model to assign greater weight to relevant metric relationships, whether local (within an instance) or global (across instances or metrics).

Each correlation type employs the above mechanism but differs in how it defines the scope of Q, K, and V (see Table 1 for specific information):

- **Corr1:** Computes interactions between features within a single instance. For each instance, the feature vector is self-attended to assess internal feature dependencies.
- **Corr2:** Models similarity between instances by averaging each instance's feature vector and applying attention across instances. This captures global instance-level patterns.
- **Corr3:** Captures the behavior of the same metric across multiple instances. Each column (metric) is treated as a vector and attention is applied across rows.
- **Corr4:** Assesses dependencies between different metrics across all instances. The attention is applied over the column-wise aggregated vectors, allowing modeling of cross-feature correlations.

These operations are further grouped into two categories based on their computation dimension:

- **Feature-oriented attention (Corr1 and Corr3):** Focuses on feature-level relationships, either within instances or across instances for the same feature.
- **Instance-oriented attention (Corr2 and Corr4):** Focuses on relationships across instances or between feature distributions aggregated over instances.

The output of each attention branch is a relation-aware representation of the input data, highlighting the correlation between metric data. These are subsequently fused using weighted aggregation, as described in the following section. This modular decomposition improves both the interpretability and flexibility of the model when applied to metric-based software defect prediction.

Feature information fusion to obtain new data In the feature information fusion stage, this study proposes a dynamic weighted fusion mechanism to deeply integrate four heterogeneous relational features (Corr1–Corr4) with raw data for generating enhanced representations. Specifically, adaptive weighting is applied to intra-instance metric relationships (Corr1), inter-instance global correlations (Corr2), intra-metric distribution patterns (Corr3), and cross-metric statistical dependencies (Corr4) through tunable coefficients $\alpha, \beta, \gamma, \delta$ (subject to $\alpha + \beta + \gamma + \delta = 1$). This constructs a comprehensive attention weight matrix:

$$\text{AvgCorr}_{ik} = \alpha \cdot \text{Corr1}_{ik} + \beta \cdot \text{Corr2}_{i} + \gamma \cdot \text{Corr3}_{ik} + \delta \cdot \text{Corr4}_{k}. \tag{7}$$

During this process, the original denominator constant 4 (which caused weight dilution) is removed, and normalization is achieved by constraining the sum of weights to 1. Additionally, the dimensional properties of different relational features are explicitly distinguished (e.g., Corr2 as instance-level vectors and Corr4 as metric-level vectors). Finally, the fused weights are applied to raw features via Hadamard product to generate new multi-granularity semantic representations:

$$\text{WeightedMetric}_{ik} = \text{AvgCorr}_{ik} \odot X_{ik}. \tag{8}$$

For software defect prediction, when the model learns $\alpha = 0.5$ and $\beta = 0.3$, the local correlations between lines of code and complexity within instances (Corr1) and the global similarity across projects (Corr2) dominate feature enhancement, while global metric relationships (Corr3/Corr4) serve as supplementary information. Through element-wise multiplication, critical features (e.g., abnormally high complexity values) are amplified and noise is suppressed, ultimately improving the model's ability to discern defect patterns. This mechanism not only achieves synergistic optimization of instance-level, metric-level, and cross-dimensional features but also provides quantifiable insights into feature importance through interpretable weight allocation.

2.6. Interpretability of MAM

2.6.1. Transparent weight allocation mechanism

MAM calculates the similarity between the query vector (Query) and the key vector (Key) to generate attention weights. These weights directly reflect the importance and mutual influence of each feature. Since the weights are computed through explicit mathematical formulas, users can clearly see the contribution of each feature in the decision-making process. These weights demonstrate the similarity between different metric features and projects in various environments. By observing these weights, users can understand which features are more important in the current project. Meanwhile, researchers can rank the features based on the weight values to perform metric selection

2.6.2. Dynamic feature selection and weighted summation

MAM enables the model to dynamically adjust feature weights based on different contexts. This allows the model to highlight important features and disregard less significant ones with varying data inputs. This dynamic feature selection improves the adaptability of the model, integrating the contribution of each feature into the final representation by weighted summation, making the influence of each feature more transparent. For example, by integrating multiple metric feature data within an instance and calculating the similarity and weighted representation between different instances, users can observe changes in the correlation between instances.

2.6.3. Multi-level relationship analysis

MAM not only considers the correlation between features and labels but also comprehensively considers the complex relationships within a instance, between instances, within a single metric feature, and between different metric features. This multi-level analysis provides a more comprehensive perspective, making the feature selection process more interpretable.

2.6.4. Visual interpretability

Attention weights can be visualized using heat maps or other visualization techniques to display the distribution of weights intuitively. This visualization further enhances the model's interpretability, making the complex feature selection process straightforward and easy to understand (Fig. 6).

3. Experiment

This section investigates the effectiveness, adaptability, and interpretability of the proposed Metric-Attention Module (MAM) across various scenarios, structured around four key research questions:

- **RQ1:** Ablation experiments are conducted to validate the effectiveness, verifying whether the proposed technical pathway and the Metric-attention module can effectively enhance the model's performance.
- **RQ2:** Does the Metric-attention module (MAM) perform well in complex task environments such as cross-project defect prediction and heterogeneous defect prediction tasks?

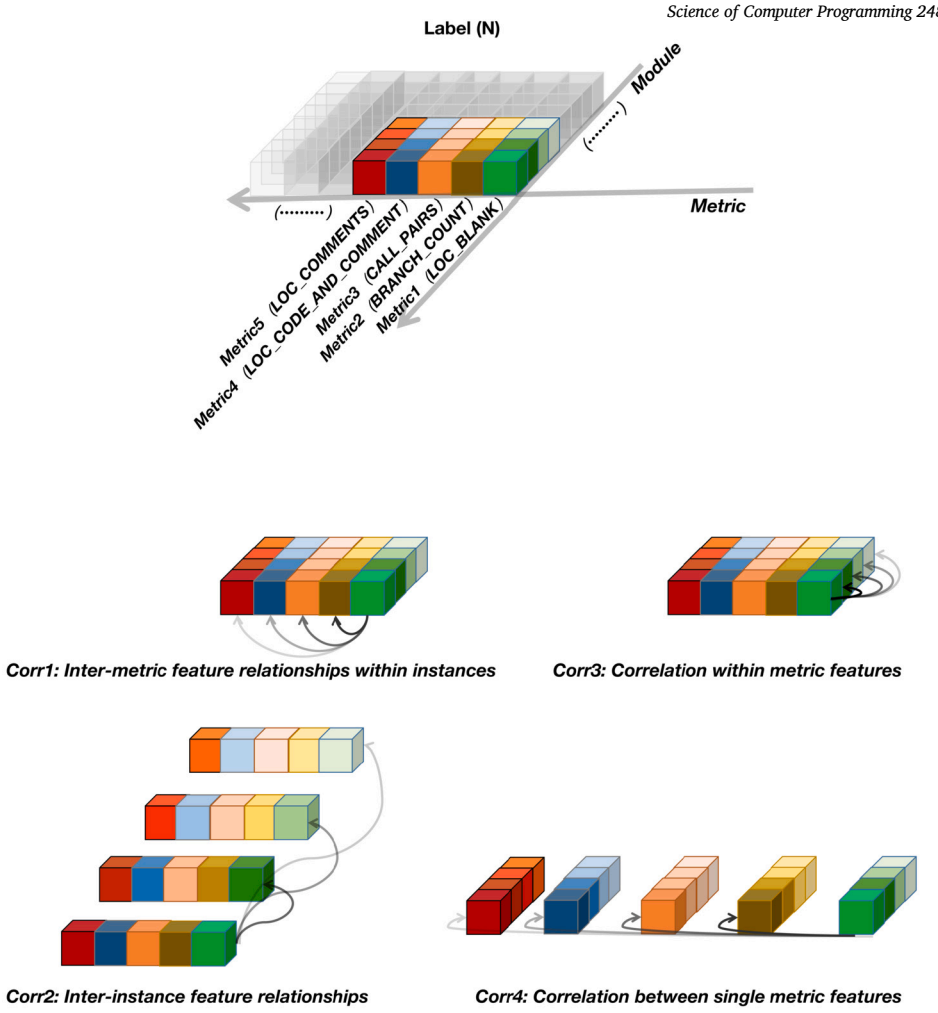


Fig. 6. Metric-Attention Module Data Processing Method.

- **RQ3:** Is the interpretability of Metric-attention module (MAM) reliable?

In **RQ1**, the effectiveness of the proposed approach is validated through experimental research. By comparing the Accuracy values before and after model enhancement under the same experimental setup, the validity of the proposed work is demonstrated. Further insights are provided through the visualization of data distributions.

In **RQ2**, the adaptability of MAM in complex task environments is explored by designing challenging task scenarios. A comprehensive evaluation of the proposed work's contribution is performed by comparing the Accuracy, F1, AUC, and MCC values of the baseline group and the MAM-enhanced group.

In **RQ3**, the interpretability of MAM is further examined using visualization methods. In the field of defect prediction, interpretable prediction models are crucial for advancing the domain. This research question aims to deepen the understanding of the interpretability process of MAM in defect prediction tasks.

3.1. Selection of datasets

In this work, we selected the AEEEM, JIRA, PROMISE, and NASA datasets for experimental evaluation and method interpretation. These datasets encompass a wide range of real-world software projects with diverse characteristics, including varying project sizes, code complexities, and defect densities, thereby ensuring comprehensive validation of the proposed approach.

AEEEM Dataset: AEEEM [31] is a dataset collection used to evaluate software engineering experimental methodologies. It contains data from software engineering projects in different domains, which can be used to validate the effectiveness of various software engineering techniques. The name AEEEM is derived from the initials of the five project names (Apache Lucene, Eclipse JDT Core, Equinox, Eclipse PDE UI, and Mylyn), and it includes 17 source code metrics, 5 historical defect metrics, 5 change entropy metrics, 17 source code entropy metrics, and 17 source code churn metrics, totaling 61 metrics.

Table 2
Summary of Datasets Used in Evaluation.

Dataset	Project	Modules	Defect Modules	Defective Ratio	Class	Granularity	Metric Features
AEEEM	EQ	324	129	39.8	Java	Class	61
AEEEM	JDT	997	206	20.7	Java	Class	61
AEEEM	LC	691	64	9.3	Java	Class	61
AEEEM	ML	1862	245	13.2	Java	Class	61
AEEEM	PDE	1497	209	14.0	Java	Class	61
AVERAGE		**1074**	**171**	**19.4**			
NASA	CM1	327	42	12.8	C	Function	38
NASA	MW1	251	25	10.0	C	Function	38
NASA	PC1	696	55	7.9	C	Function	38
NASA	PC3	1073	132	12.3	C	Function	38
NASA	PC5	1276	176	13.8	C	Function	38
AVERAGE		**725**	**86**	**11.4**			
JIRA	activemg-5.0.0	1884	293	15.6	Java	File	65
JIRA	derby-10.5.1.1	2705	383	14.2	Java	File	65
JIRA	groovy-1.6.0.betal	821	70	8.5	Java	File	65
JIRA	hbase-0.94.0	1059	218	20.6	Java	File	65
JIRA	hive-0.9.0	1416	283	20.0	Java	File	65
JIRA	jruby-1.1	731	87	11.9	Java	File	65
JIRA	wicket-1.3.0.beta2	1763	130	7.4	Java	File	65
AVERAGE		**1483**	**209**	**14.0**			
PROMISE	ant-1.7	745	166	22.3	Java	Class	20
PROMISE	camel-1.4	872	145	16.6	Java	Class	20
PROMISE	ivy-2.0	352	40	11.4	Java	Class	20
PROMISE	jedit-4.0	306	75	24.5	Java	Class	20
PROMISE	log4j-1.0	135	34	25.2	Java	Class	20
PROMISE	poi-2.0	314	37	11.8	Java	Class	20
PROMISE	tomcat-6.0	858	77	9.0	Java	Class	20
PROMISE	velocity-1.6	229	78	34.1	Java	Class	20
PROMISE	xalan-2.4	723	110	15.2	Java	Class	20
PROMISE	xerces-1.3	453	69	12.2	Java	Class	20
AVERAGE		**499**	**93**	**18.4**			

JIRA Dataset: JIRA [32] is a widely used issue tracking and project management tool. The JIRA dataset is collected from the JIRA system and typically includes defect reports, issue tracking, task assignments, and other information. This data can be used to analyze and understand the issues and defects in software projects. The dataset contains project modules representing Java files with 65 metrics, including 54 code metrics, 5 process metrics, and 6 ownership metrics.

PROMISE Dataset [33]: The PROMISE project provides a series of datasets for software engineering research. These datasets include data on software defect prediction, software quality assessment, software metrics, and other areas, and can be used to evaluate and compare various software engineering techniques and methods. The full PROMISE dataset contains 48 versions from 15 open-source projects, 27 versions from 6 proprietary projects, and 17 versions from 17 academic projects implemented by students. The project modules in the dataset are represented by 20 metrics, including McCabe complexity, CK metrics, and other object-oriented metrics.

NASA Dataset [24]: The NASA dataset typically refers to datasets provided by NASA's Software Engineering Laboratory. These datasets contain data collected during the software development and maintenance process of NASA projects. The data are used in software engineering research, such as software defect prediction and software metrics. The dataset includes a range of NASA development projects, including satellite instruments, ground control systems, and some flight control modules. Over the years, the NASA dataset has been widely used for performance evaluation in defect prediction methods, with metrics such as module size, readability, complexity, and other static code metrics.

These four benchmark datasets—AEEEM, JIRA, PROMISE, and NASA—are widely used in software engineering and software defect prediction literature. They have undergone standardized collection and curation, ensuring high reliability and consistent structure, which facilitates reproducibility and comparability across studies. Moreover, they span multiple programming languages (e.g., Java, C), defect granularities (file-, class-, and function-level), and project types (industrial and open-source). This diversity makes them representative of a broad range of practical defect prediction scenarios. Detailed dataset statistics are provided in Table 2.

3.2. Selection of evaluation indicators

In order to comprehensively consider the accuracy, coverage, and overall performance of the classifier, and to fully suit tasks such as software defect prediction that require high accuracy and robustness, the authors have selected F-1, AUC, and MCC as the evaluation metrics for this experiment.

Accuracy: Represents the ratio between the number of correctly predicted samples and the total number of samples. This metric is used to assess the performance of a classification model. A high accuracy means the model is able to correctly predict the majority of the samples.

F-1 Score: The F-1 score is the harmonic mean of precision and recall, which balances the accuracy and coverage of the classifier. In software defect detection, it is important to ensure that true defects are detected as accurately as possible while minimizing false positives. Therefore, the F-1 score is a crucial evaluation metric.

AUC (Area Under the ROC Curve): The Area Under the ROC Curve (AUC) measures the performance of the classifier at different thresholds. It is robust to the imbalance between classes. In software defect detection, there is often a severe imbalance between negative samples (non-defects) and positive samples (defects), making AUC a comprehensive metric that evaluates performance without being influenced by data imbalance.

MCC (Matthews Correlation Coefficient): MCC is a performance metric that considers true positives, false positives, true negatives, and false negatives. It provides a more balanced evaluation, particularly in imbalanced datasets. The MCC ranges from -1 to 1, where 1 indicates perfect prediction, 0 indicates random prediction, and -1 indicates completely opposite predictions. In software defect detection, MCC helps evaluate the overall performance of the classifier while accounting for the impact of imbalanced datasets.

3.3. Wilcoxon signed-rank test for statistical significance

To assess whether the performance improvements of our proposed model are statistically significant, we adopt the Wilcoxon Signed-Rank Test [40], a widely used non-parametric test for paired samples. Unlike the paired t-test, which assumes a normal distribution of differences, the Wilcoxon test makes no such assumptions, making it especially suitable for evaluating machine learning models across multiple datasets. In our case, the Wilcoxon test is used to compare the performance of MAM-enhanced models with their original counterparts across key metrics such as F1 Score, AUC, and MCC. A low p-value (typically < 0.05) indicates that the observed improvements are unlikely to have occurred by chance, thereby affirming the statistical significance of MAM's performance gains.

3.4. Effectiveness of the metric-attention module in the Within-Project Defect Prediction (WPDP) task model

The purpose of this experiment is to validate the effectiveness of the proposed Metric-attention Module (MAM) in the context of within-project defect prediction (WPDP). The experimental design adheres to the principle of using generic datasets and models for evaluation. This is because high-quality datasets or complex models could directly impact performance, making it difficult to determine whether the performance improvement is attributable to the inclusion of the Metric-attention Module.

For dataset selection, the authors randomly selected 10 projects from the chosen datasets, namely AEEEM, JIRA, PROMISE, and NASA, for testing. For model selection, they included random forest, logistic regression, decision tree to provide a more comprehensive evaluation of the work's quality.

In all experiments, the training and testing datasets were split proportionally (e.g., 80:20). Furthermore, the experiments employed 30-repetition holdout validation to reduce statistical bias and variance. Finally, the effectiveness of the Metric-attention Module in WPDP tasks was assessed by comparing the prediction performance metrics of the models before and after the module's integration.

3.5. Effectiveness of the metric-attention module in cross-project and heterogeneous defect prediction tasks

Cross-project defect prediction (CPDP) has emerged as a critical direction to address the challenge of acquiring software engineering datasets and is an indispensable part of software defect prediction research. Despite significant contributions by many researchers, CPDP still faces issues such as limited performance and weak reproducibility [30]. Similarly, heterogeneous defect prediction (HDP), as a special case of CPDP, encounters even greater challenges. CPDP is particularly applicable in scenarios where the target project is new or has only a small amount of training data available.

Previous CPDP studies have primarily focused on leveraging training data from other projects (i.e., source projects) and employing transfer learning techniques to reduce distribution differences between projects. However, practitioners often use different metrics to measure program modules across projects, making CPDP with heterogeneous metrics—known as HDP—more challenging. This underscores the necessity of utilizing existing data and enhancing current models to tackle defect detection challenges effectively.

To address these issues, this work introduces the Metric-attention Module (MAM), which leverages existing data resources and models to improve performance, thereby alleviating problems such as limited performance and data scarcity. HDP tasks, as adjustable scenarios within CPDP, were chosen as the focus of this study. The primary objectives are to validate the effectiveness of the proposed MAM in cross-project and heterogeneous defect prediction tasks and to evaluate its adaptability across various models.

In this study, four classic HDP models were selected as baseline models:

- **JMST [34]**: A heterogeneous defect detection method that combines metric selection and transformation to reduce heterogeneity between source and target projects through a mapping matrix.
- **CCA+ [35]**: An HDP method that does not incorporate target project data, used to evaluate whether the inclusion of target data enhances performance.
- **CLSUP [36]**: An HDP method designed to handle class imbalance issues, validating its effectiveness in scenarios with imbalanced data distributions.

- **CPDP-IFS [37]**: A high-performance HDP method used to test MAM's adaptability to complex and diverse models.

The experimental design involved constructing prediction combinations by selecting a target project from one dataset and a source project from one of the other two, ensuring heterogeneity. This approach allowed for the creation of 141 prediction combinations across 10 projects from the three datasets. The effectiveness of MAM was evaluated by comparing the performance indicators of the baseline models with and without MAM integration. Improvement in these indicators would confirm MAM's effectiveness.

In terms of experimental evaluation, since the ultimate goal of this experiment is to verify the effectiveness of the proposed approach and module, the evaluation criteria consider an improvement in various performance metrics compared to the baseline model, under different models and task environments, as evidence of effectiveness.

3.5.1. Sensitivity to model complexity

To investigate whether the performance gains attributed to the Metric Attention Module (MAM) are influenced by the complexity of the base model, we conduct a sensitivity experiment under varying model depths. The goal is to determine whether MAM's improvements stem from its feature enhancement capability rather than the inherent capacity of the base model. In this experiment, we adopt a simple Multi-Layer Perceptron (MLP) as the classifier, varying its depth (i.e., the number of hidden layers: 1, 2, and 3) while keeping other configurations constant. Each MLP configuration includes standard fully connected layers with ReLU activations. The experimental setup follows a Within-Project Defect Prediction (WPDP) scenario using the CM1 dataset. All data are standardized, and consistent train-test splits and evaluation metrics are used across settings. This experiment aims to answer the following research questions:

- Does MAM's performance enhancement depend on the expressive capacity of the base model?
- Can MAM improve performance even under low-complexity model conditions?

3.5.2. Interpretable options for the metric-attention module

To verify the interpretability of the Metric-Attention Module (MAM), we trained the model as described in $Section 2.5$ and provided a visual interpretation of the experimental results. The accuracy of the interpreted features was validated through the following experimental design:

The experiment consists of five groups:

- Baseline Group: Uses the original data without metric selection as the control.
- Experiment 1: Selects the top 20 metric features based on weight values.
- Experiment 2: Selects the top 10 metric features based on weight values.
- Experiment 3: Selects the top 5 metric features based on weight values.
- Experiment 4: Selects the top 2 metric features based on weight values.
- Experiment 5: Selects the top 1 metric features based on weight values.

To facilitate rapid validation, we used the CM1 file from the NASA dataset for model training, with a logistic regression model chosen for clearer visual interpretation and reduced experimental complexity. At the same time, we selected ReliefF as a comparison group to enable a clearer contrast of interpretability results. ReliefF [39] was selected as the baseline because it is a classical and widely used feature selection method that provides explicit importance scores for each feature, offering a degree of interpretability. By comparing it with MAM's hierarchical attention mechanism, we can effectively assess MAM's advantages in identifying feature importance and enhancing model interpretability.

The effectiveness of MAM is evaluated based on its ability to analyze the importance of metric features and provide interpretable selections. The selected features should influence final model training outcomes, with Accuracy, F1, MCC, and AUC used as evaluation criteria:

- Improvement in Indicators: If performance improves after reducing metric features, it suggests the removed features were redundant, negatively impacting learning efficiency and model performance.
- No Change in Indicators: If performance remains consistent with the baseline, the method is still effective, as fewer metrics were used, indicating that the selected features effectively represent project information and improve model efficiency.

Through these evaluations, we assess the interpretability and effectiveness of MAM in selecting and analyzing important metric features.

4. Results

4.1. Results of ablation experiments in the Within-project Defect Prediction (WPDP) task model

The prediction of software defects within-project has always been an important research direction in the field of software defect prediction. This direction faces two practical issues.

Table 3
Performance Comparison between MAM-Processed and Unprocessed Decision Tree Models.

Project	MAM Processed				Unprocessed				Improvement			
	Acc	F1	AUC	MCC	Acc	F1	AUC	MCC	ΔAcc	ΔF1	ΔAUC	ΔMCC
Lucene	0.999	0.999	0.993	0.992	0.868	0.874	0.632	0.236	0.131	0.125	0.361	0.756
Mylyn	0.999	0.999	0.997	0.996	0.819	0.822	0.632	0.238	0.181	0.178	0.364	0.759
hbase-0.94.0	1.000	1.000	1.000	1.000	0.788	0.789	0.688	0.370	0.212	0.211	0.312	0.630
ant-1.7	0.999	0.999	0.999	0.998	0.748	0.752	0.651	0.295	0.251	0.247	0.347	0.703
JDT	1.000	1.000	0.999	0.999	0.806	0.807	0.712	0.416	0.194	0.192	0.287	0.583
groovy-1_6_BETA_1	1.000	1.000	1.000	1.000	0.906	0.910	0.731	0.421	0.094	0.090	0.269	0.579
tomcat	1.000	1.000	1.000	1.000	0.864	0.866	0.610	0.209	0.136	0.134	0.390	0.791
jruby-1.1	0.999	0.999	0.997	0.997	0.876	0.876	0.715	0.422	0.123	0.123	0.282	0.574
camel-1.4	1.000	1.000	0.999	0.999	0.753	0.758	0.577	0.149	0.246	0.242	0.422	0.849
poi-2.0	0.999	0.999	0.996	0.995	0.855	0.859	0.683	0.331	0.144	0.140	0.313	0.664
jedit-4.0	1.000	1.000	1.000	1.000	0.747	0.746	0.665	0.333	0.253	0.254	0.335	0.667
xalan-2.4	1.000	1.000	1.000	1.000	0.794	0.795	0.615	0.222	0.206	0.205	0.385	0.778
Average	**1.000**	**1.000**	**0.998**	**0.998**	**0.819**	**0.821**	**0.659**	**0.304**	**0.181**	**0.178**	**0.339**	**0.694**

Table 4
Performance Comparison between MAM-Processed and Unprocessed Logistic Regression Models.

Project	MAM Processed				Unprocessed				Improvement			
	Acc	F1	AUC	MCC	Acc	F1	AUC	MCC	ΔAcc	ΔF1	ΔAUC	ΔMCC
Lucene	0.999	0.999	1.000	0.994	0.914	0.903	0.747	0.335	0.085	0.096	0.253	0.659
Mylyn	1.000	1.000	1.000	1.000	0.869	0.843	0.772	0.261	0.131	0.157	0.228	0.739
hbase-0.94.0	1.000	1.000	1.000	1.000	0.832	0.819	0.836	0.436	0.168	0.181	0.164	0.564
ant-1.7	1.000	1.000	1.000	0.998	0.823	0.803	0.817	0.417	0.176	0.197	0.183	0.581
JDT	0.999	0.999	1.000	0.998	0.859	0.848	0.836	0.523	0.140	0.151	0.164	0.475
groovy-1_6_BETA_1	1.000	1.000	1.000	1.000	0.928	0.918	0.858	0.403	0.072	0.082	0.142	0.597
tomcat	1.000	1.000	1.000	1.000	0.906	0.885	0.824	0.235	0.094	0.115	0.176	0.765
jruby-1.1	1.000	1.000	1.000	1.000	0.907	0.899	0.875	0.514	0.093	0.101	0.125	0.486
camel-1.4	0.998	0.998	0.998	0.993	0.826	0.777	0.709	0.138	0.173	0.221	0.289	0.856
poi-2.0	1.000	1.000	1.000	1.000	0.876	0.845	0.698	0.175	0.124	0.155	0.302	0.825
jedit-4.0	1.000	1.000	1.000	1.000	0.802	0.783	0.793	0.421	0.198	0.217	0.207	0.579
xalan-2.4	0.999	0.999	1.000	0.996	0.847	0.815	0.761	0.239	0.152	0.184	0.239	0.757
Average	**1.000**	**1.000**	**1.000**	**0.998**	**0.866**	**0.845**	**0.794**	**0.341**	**0.134**	**0.155**	**0.206**	**0.657**

Table 5
Performance Comparison between MAM-Processed and Unprocessed Random Forest Models.

Project	MAM Processed				Unprocessed				Improvement			
	Acc	F1	AUC	MCC	Acc	F1	AUC	MCC	ΔAcc	ΔF1	ΔAUC	ΔMCC
Lucene	1.000	1.000	1.000	1.000	0.924	0.905	0.829	0.356	0.076	0.095	0.171	0.644
Mylyn	1.000	1.000	1.000	1.000	0.875	0.851	0.835	0.307	0.125	0.149	0.165	0.693
hbase-0.94.0	1.000	1.000	1.000	1.000	0.840	0.827	0.864	0.463	0.160	0.173	0.136	0.537
ant-1.7	1.000	1.000	1.000	1.000	0.825	0.814	0.835	0.446	0.175	0.186	0.165	0.554
JDT	1.000	1.000	1.000	1.000	0.860	0.848	0.882	0.523	0.140	0.152	0.118	0.477
groovy-1_6_BETA_1	1.000	1.000	1.000	1.000	0.946	0.937	0.890	0.560	0.054	0.063	0.110	0.440
tomcat	1.000	1.000	1.000	1.000	0.909	0.882	0.829	0.229	0.091	0.118	0.171	0.771
jruby-1.1	1.000	1.000	1.000	1.000	0.925	0.918	0.881	0.607	0.075	0.082	0.119	0.393
camel-1.4	1.000	1.000	1.000	0.999	0.843	0.804	0.703	0.269	0.157	0.195	0.297	0.730
poi-2.0	1.000	1.000	1.000	1.000	0.888	0.867	0.830	0.295	0.112	0.133	0.170	0.705
jedit-4.0	1.000	1.000	1.000	1.000	0.817	0.805	0.819	0.477	0.183	0.195	0.181	0.523
xalan-2.4	1.000	1.000	1.000	1.000	0.847	0.815	0.802	0.238	0.153	0.185	0.198	0.762
Average	**1.000**	**1.000**	**1.000**	**1.000**	**0.875**	**0.856**	**0.833**	**0.398**	**0.125**	**0.144**	**0.167**	**0.602**

Effectiveness of MAM in WPDP tasks From the results (see Tables 3–5), MAM processing led to substantial performance gains across all three models. In 30 experimental settings, average improvements were observed as follows: Accuracy +14.7%, F1-Score +15.9%, AUC +23.7%, and MCC +65.1%. These findings suggest that MAM significantly enhances defect prediction, even for lightweight classifiers such as Logistic Regression and Decision Trees.

Statistical confidence and practical implications The Wilcoxon signed-rank tests (Table 6) confirm the statistical significance of these gains ($p < 0.0001$ across all metrics). For example, MCC improvements reached 0.695 (Decision Tree), 0.657 (Logistic Regression), and 0.602 (Random Forest). MAM achieves this by capturing both local and global relational structures through multi-attention encoding,

Table 6
Wilcoxon test results with 95% CIs: MAM vs. original models.

Model	Metric	MAM (95% CI)	Original (95% CI)	Imp.	p-value
RF	Accuracy	1.000 [1.000, 1.000]	0.875 [0.844, 0.905]	0.125	<0.0001
RF	F1-Score	1.000 [0.999, 1.000]	0.856 [0.819, 0.892]	0.144	<0.0001
RF	AUC	1.000 [1.000, 1.000]	0.834 [0.790, 0.878]	0.166	<0.0001
RF	MCC	1.000 [0.999, 1.000]	0.398 [0.346, 0.449]	0.602	<0.0001
DT	Accuracy	1.000 [0.999, 1.000]	0.819 [0.777, 0.861]	0.181	<0.0001
DT	F1-Score	1.000 [0.999, 1.000]	0.821 [0.783, 0.859]	0.178	<0.0001
DT	AUC	0.999 [0.998, 1.000]	0.659 [0.610, 0.708]	0.339	<0.0001
DT	MCC	0.998 [0.996, 0.999]	0.304 [0.247, 0.361]	0.695	<0.0001
LR	Accuracy	1.000 [1.000, 1.000]	0.866 [0.832, 0.900]	0.134	<0.0001
LR	F1-Score	1.000 [0.999, 1.000]	0.845 [0.804, 0.886]	0.155	<0.0001
LR	AUC	1.000 [1.000, 1.000]	0.794 [0.739, 0.849]	0.206	<0.0001
LR	MCC	0.998 [0.996, 1.000]	0.341 [0.279, 0.404]	0.657	<0.0001

helping models better suppress noise and extract relevant features. However, in small datasets, extremely high validation scores may reflect overfitting due to limited sample size after splitting. Thus, while MAM shows strong general potential, its application in real-world scenarios should consider dataset size and model capacity.

4.2. Effective performance test results in heterogeneous defect prediction (HDP) tasks

Overall model performance Across 188 experimental groups (see Tables 6–9), all four models—CCA+, CLSUP, JMST, and CPDPIFS—exhibited varying degrees of improvement. Notably, in the CCA+ model, the average improvements in F1, AUC, and MCC were 29%, 16%, and 38%, respectively, indicating a substantial enhancement over the baseline. Similar trends were observed in CLSUP, CPDPIFS, and JMST. For instance, in CCA+, experimental groups like *ant-1.7_to_JDT*, *camel-1.4_to_JDT*, and *JDT_to_ant-1.7* achieved near-perfect values across all three metrics. CLSUP and CPDPIFS also demonstrated robust performance on selected tasks.

Performance degradation cases Despite these improvements, performance was suboptimal in a few projects. For example, the JMST model underperformed on *ant-1.7_to_Mylyn* and *camel-1.4_to_Mylyn*; CCA+ struggled with *Lucene_to_xalan-2.4* and *JDT_to_xalan-2.4*; CLSUP showed weaker results on *ant-1.7_to_PDE* and *camel-1.4_to_PDE*; and CPDPIFS failed in *ant-1.7_to_Mylyn* and *camel-1.4_to_Mylyn*. These failures were mainly associated with the Mylyn and xalan datasets, which suffer from small data size, class imbalance, and label noise. As noted by Lessmann et al. [38], such noisy labels can severely affect model performance (Table 10).

Failure statistics and data quality impact Among the 564 metric values across 188 groups, only 90 metrics (15.9%) in 37 experimental groups (19.6%) showed performance degradation. Notably, groups involving “Mylyn” and “xalan-2.4” accounted for 49% and 35% of all declines, respectively—together contributing 84% of the underperforming cases. This demonstrates that while MAM generally improves performance, its effectiveness is highly dependent on data quality.

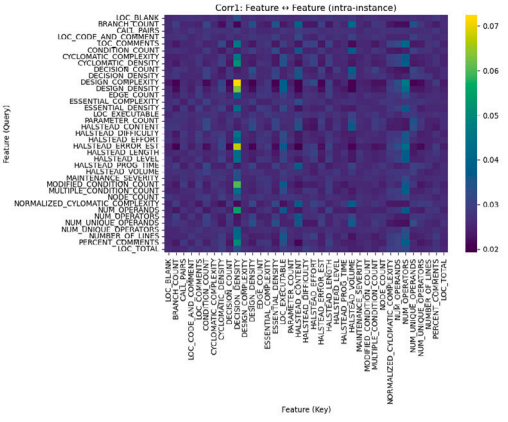
Model-wise comparison From the model architecture perspective, CCA+ achieved the largest improvement margins, with F1, AUC, and MCC gains reaching 59%, 42%, and 83%, respectively. CLSUP also showed significant boosts (up to 54%, 20%, and 63%), while CPDPIFS and JMST exhibited more moderate gains. Models with stronger learning capability (e.g., CCA+, CLSUP) consistently benefited more from MAM, while weaker models like CPDPIFS struggled, particularly on noisy or imbalanced datasets like *jedit-4.0* and *Mylyn*.

Summary and significance testing Overall, MAM proves to be effective in HDP tasks, although its performance is less stable compared to within-project settings due to domain shifts, label noise, and class imbalance. The baseline model’s capability also influences MAM’s impact.

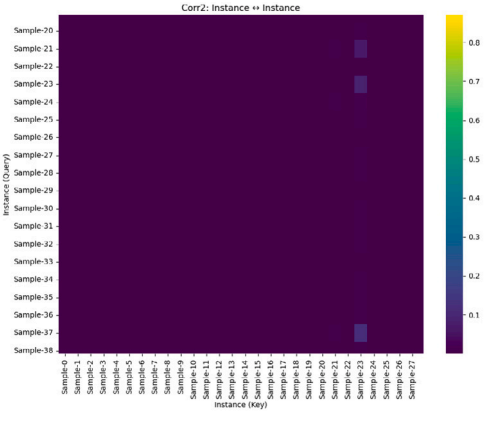
To assess the statistical significance of these improvements, we conducted Wilcoxon signed-rank tests across all metrics. Results show that performance improvements are statistically significant ($p < 0.05$) for most metrics and models, particularly in F1-score and MCC for CCA+, CLSUP, CPDPIFS, and JMST, validating the robustness of MAM’s effectiveness in cross-project scenarios (Table 11).

4.3. The result of sensitivity to model complexity

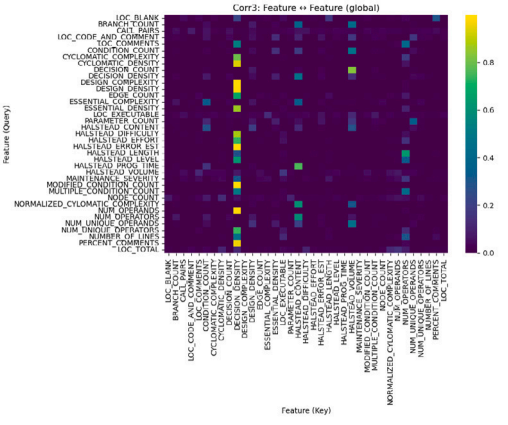
The results (see Table 12) demonstrate that MAM consistently enhances model performance across all tested depths. Notably, the improvements are more pronounced with shallower MLPs, suggesting that MAM’s effectiveness derives primarily from its ability to enrich feature representations rather than relying on the complexity of the classifier itself. These findings support the robustness and generalizability of the proposed MAM architecture.



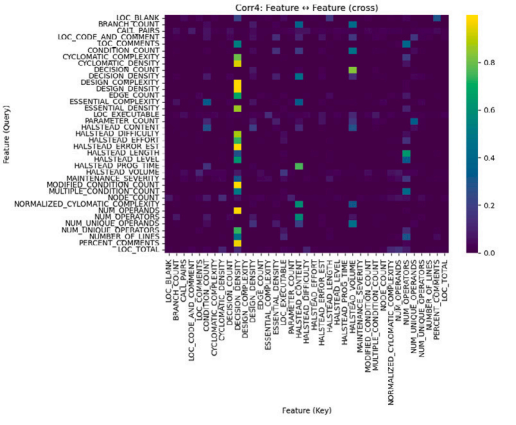
(a) Corr1: Intra-instance feature attention



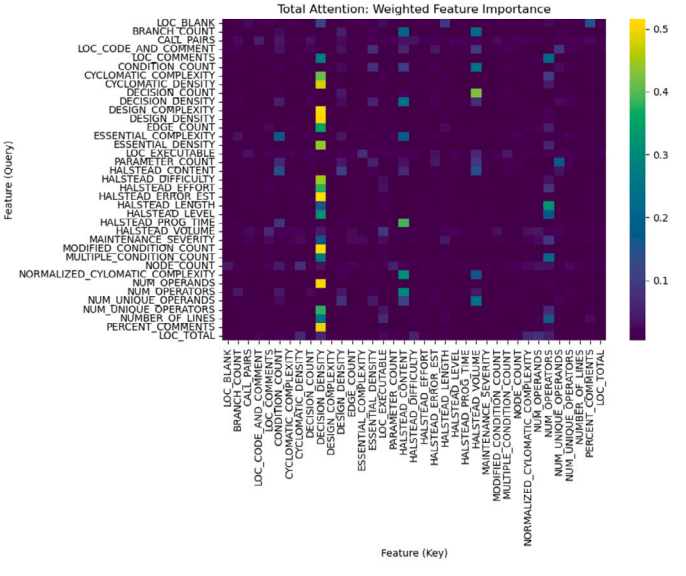
(b) Corr2: Inter-instance frature attention



(c) Corr3: Within metric feature attention



(d) Corr4: Between single metric feature attention



(e) Total Attention: Aggregated Feature Importance

Fig. 7. Visualization of the attention weights across four attention submodules and the aggregated importance. Corr1–4 capture intra-instance, inter-instance, within metric feature attention and between single metric feature attention.

Table 7
Performance on CCA+ model.

CCA+	Processed			Unprocessed			Improve		
	F_1	AUC	MCC	F_1	AUC	MCC	F_1	AUC	MCC
ant-1.7_to_JDT	0.99	1	0.99	0.57	0.77	0.46	**0.42**	**0.23**	**0.54**
ant-1.7_to_Lucene	0.95	1	0.95	0.37	0.8	0.32	**0.58**	**0.2**	**0.63**
ant-1.7_to_Mylyn	0.29	0.71	0.15	0.35	0.71	0.24	-0.07	0	-0.09
ant-1.7_to_PDE	0.53	0.88	0.47	0.37	0.7	0.25	**0.17**	**0.18**	**0.22**
camel-1.4_to_JDT	1	1	1	0.59	0.8	0.46	**0.41**	**0.2**	**0.53**
camel-1.4_to_Lucene	0.95	1	0.95	0.36	0.79	0.32	**0.59**	**0.21**	**0.63**
camel-1.4_to_Mylyn	0.36	0.71	0.26	0.36	0.71	0.25	0	0	**0.01**
camel-1.4_to_PDE	0.51	0.87	0.45	0.37	0.71	0.25	**0.14**	**0.16**	**0.2**
JDT_to_ant-1.7	1	1	1	0.49	0.72	0.27	**0.51**	**0.28**	**0.73**
JDT_to_camel-1.4	0.92	0.99	0.9	0.43	0.73	0.29	**0.49**	**0.26**	**0.61**
JDT_to_jedit-4.0	0.83	0.97	0.78	0.57	0.75	0.4	**0.25**	**0.22**	**0.39**
JDT_to_tomcat	0.99	1	0.99	0.42	0.88	0.41	**0.56**	**0.12**	**0.57**
JDT_to_velocity-1.6	0.98	1	0.97	0.44	0.58	0.17	**0.54**	**0.42**	**0.8**
JDT_to_xalan-2.4	0.13	0.69	-0.04	0.39	0.66	0.2	-0.26	**0.03**	-0.24
jedit-4.0_to_JDT	0.95	0.96	0.91	0.57	0.78	0.46	**0.38**	**0.18**	**0.45**
jedit-4.0_to_Lucene	0.82	0.99	0.82	0.36	0.78	0.32	**0.46**	**0.21**	**0.5**
jedit-4.0_to_Mylyn	0.4	0.76	0.32	0.34	0.66	0.22	**0.06**	**0.09**	**0.09**
jedit-4.0_to_PDE	0.52	0.88	0.45	0.35	0.67	0.21	**0.17**	**0.22**	**0.24**
Lucene_to_ant-1.7	1	1	1	0.53	0.77	0.39	**0.47**	**0.23**	**0.6**
Lucene_to_camel-1.4	0.83	0.98	0.8	0.4	0.73	0.27	**0.42**	**0.25**	**0.53**
Lucene_to_jedit-4.0	0.68	0.83	0.53	0.61	0.8	0.47	**0.06**	**0.04**	**0.07**
Lucene_to_tomcat	1	1	1	0.43	0.86	0.41	**0.57**	**0.14**	**0.59**
Lucene_to_velocity-1.6	0.81	0.82	0.63	0.38	0.58	0.13	**0.42**	**0.24**	**0.5**
Lucene_to_xalan-2.4	0.15	0.59	-0.03	0.51	0.82	0.43	-0.36	-0.23	-0.46
Mylyn_to_ant-1.7	0.91	0.91	0.83	0.52	0.74	0.33	**0.38**	**0.16**	**0.5**
Mylyn_to_camel-1.4	0.88	0.95	0.85	0.45	0.76	0.33	**0.43**	**0.2**	**0.52**
Mylyn_to_jedit-4.0	0.77	0.93	0.7	0.54	0.77	0.38	**0.23**	**0.15**	**0.32**
Mylyn_to_tomcat	0.98	1	0.98	0.42	0.89	0.42	**0.55**	**0.11**	**0.56**
Mylyn_to_velocity-1.6	0.98	0.99	0.97	0.45	0.64	0.18	**0.53**	**0.36**	**0.79**
Mylyn_to_xalan-2.4	0.28	0.78	0.14	0.49	0.82	0.41	-0.21	-0.04	-0.26
PDE_to_ant-1.7	0.9	0.9	0.81	0.54	0.77	0.39	**0.36**	**0.13**	**0.42**
PDE_to_camel-1.4	0.91	0.99	0.9	0.44	0.72	0.3	**0.47**	**0.27**	**0.59**
PDE_to_jedit-4.0	0.75	0.91	0.64	0.58	0.75	0.4	**0.16**	**0.16**	**0.24**
PDE_to_tomcat	1	1	1	0.41	0.85	0.39	**0.59**	**0.15**	**0.61**
PDE_to_velocity-1.6	0.98	0.99	0.98	0.41	0.6	0.14	**0.58**	**0.4**	**0.83**
PDE_to_xalan-2.4	0.18	0.65	0.03	0.49	0.82	0.39	-0.31	-0.17	-0.36
tomcat_to_JDT	1	1	1	0.58	0.8	0.47	**0.41**	**0.2**	**0.53**
tomcat_to_Lucene	0.95	1	0.95	0.37	0.79	0.32	**0.59**	**0.21**	**0.63**
tomcat_to_Mylyn	0.4	0.75	0.32	0.35	0.71	0.23	**0.05**	**0.04**	**0.09**
tomcat_to_PDE	0.53	0.87	0.47	0.37	0.72	0.24	**0.16**	**0.15**	**0.23**
velocity-1.6_to_JDT	1	1	1	0.58	0.79	0.46	**0.42**	**0.21**	**0.53**
velocity-1.6_to_Lucene	0.79	0.99	0.79	0.35	0.78	0.3	**0.44**	**0.22**	**0.5**
velocity-1.6_to_Mylyn	0.47	0.82	0.42	0.37	0.73	0.26	**0.1**	**0.09**	**0.16**
velocity-1.6_to_PDE	0.5	0.85	0.44	0.37	0.71	0.25	**0.13**	**0.14**	**0.18**
xalan-2.4_to_JDT	1	1	0.99	0.59	0.78	0.47	**0.41**	**0.21**	**0.52**
xalan-2.4_to_Lucene	0.66	0.97	0.66	0.36	0.79	0.32	**0.3**	**0.18**	**0.35**
xalan-2.4_to_Mylyn	0.32	0.69	0.19	0.34	0.67	0.22	-0.02	**0.02**	-0.03
xalan-2.4_to_PDE	0.53	0.88	0.47	0.37	0.71	0.25	**0.16**	**0.18**	**0.22**
Average	**0.73**	**0.91**	**0.68**	**0.44**	**0.75**	**0.32**	**0.29**	**0.16**	**0.36**

4.4. Metric-attention module interpretation results

4.4.1. Weight visualization effect

The depth of the color in the visualization indicates the magnitude of the weight, with colors ranging from light to dark. Darker colors represent greater weights, as shown in the legend that details the range of weight values. By examining the color depth, we can gauge the importance of various features in the model.

The X-axis and Y-axis correspond to feature indices, where the 0th position on both axes represents the first feature. Each cell in the grid shows the weight between two features, indicating how the feature on the X-axis influences the feature on the Y-axis.

To determine feature importance, look for cells with the darkest colors, which signify higher weights. For example, a dark intersection at the 20th row and 20th column highlights the significant importance of the 20th feature in the attention mechanism. Conversely, lighter areas denote features with lower weights and thus less impact on the model.

Dark intersections between certain features suggest a strong relationship or mutual influence. For example, in Fig. 7(a), the highlighted region between DECISION_DENSITY and DESIGN_COMPLEXITY indicates a strong feature correlation. By examining the subfigures in Fig. 7, we observe that DECISION_DENSITY consistently shows strong associations with multiple features under different

Table 8
Performance on CLSUP model.

CLSUP	Processed			Unprocessed			Improve		
	F_1	AUC	MCC	F_1	AUC	MCC	F_1	AUC	MCC
ant-1.7_to_JDT	1	1	1	0.84	0.99	0.8	**0.16**	**0.01**	**0.2**
ant-1.7_to_Lucene	0.76	1	0.76	0.52	0.99	0.54	**0.24**	**0.01**	**0.22**
ant-1.7_to_Mylyn	0.83	1	0.81	0.71	1	0.7	**0.11**	0	**0.12**
ant-1.7_to_PDE	0.72	0.99	0.69	0.69	1	0.68	**0.02**	-0.01	**0.02**
camel-1.4_to_JDT	1	1	1	0.86	1	0.83	**0.14**	0	**0.17**
camel-1.4_to_Lucene	0.76	1	0.76	0.53	0.99	0.54	**0.23**	**0.01**	**0.22**
camel-1.4_to_Mylyn	0.83	1	0.82	0.74	1	0.72	**0.09**	0	**0.1**
camel-1.4_to_PDE	0.72	0.99	0.7	0.72	1	0.7	0	-0.01	0
JDT_to_ant-1.7	1	1	1	0.87	1	0.84	**0.13**	0	**0.16**
JDT_to_camel-1.4	0.99	0.99	0.99	0.91	1	0.89	**0.08**	-0.01	**0.1**
JDT_to_jedit-4.0	0.95	1	0.93	0.79	0.98	0.74	**0.16**	**0.02**	**0.2**
JDT_to_tomcat	1	1	1	0.61	1	0.62	**0.39**	0	**0.38**
JDT_to_velocity-1.6	0.99	1	0.99	0.81	0.96	0.7	**0.19**	**0.04**	**0.29**
JDT_to_xalan-2.4	0.9	0.99	0.89	0.76	1	0.74	**0.13**	-0.01	**0.15**
jedit-4.0_to_JDT	1	1	1	0.82	0.99	0.78	**0.18**	**0.01**	**0.22**
jedit-4.0_to_Lucene	0.77	1	0.77	0.5	0.99	0.51	**0.27**	**0.01**	**0.26**
jedit-4.0_to_Mylyn	0.83	1	0.82	0.68	1	0.67	**0.15**	0	**0.15**
jedit-4.0_to_PDE	0.72	0.99	0.7	0.66	1	0.64	**0.06**	-0.01	**0.06**
Lucene_to_ant-1.7	1	1	1	0.82	0.99	0.78	**0.18**	**0.01**	**0.22**
Lucene_to_camel-1.4	0.99	0.99	0.99	0.87	0.99	0.85	**0.13**	0	**0.15**
Lucene_to_jedit-4.0	0.95	1	0.93	0.64	0.91	0.52	**0.31**	**0.09**	**0.41**
Lucene_to_tomcat	1	1	1	0.68	1	0.69	**0.32**	0	**0.31**
Lucene_to_velocity-1.6	0.99	1	0.99	0.69	0.88	0.5	**0.3**	**0.12**	**0.49**
Lucene_to_xalan-2.4	0.91	0.99	0.9	0.75	0.99	0.72	**0.17**	**0.01**	**0.18**
Mylyn_to_ant-1.7	1	1	1	0.62	0.93	0.52	**0.38**	**0.07**	**0.48**
Mylyn_to_camel-1.4	0.99	0.99	0.99	0.58	0.95	0.51	**0.42**	**0.05**	**0.48**
Mylyn_to_jedit-4.0	0.96	1	0.94	0.56	0.83	0.4	**0.39**	**0.17**	**0.54**
Mylyn_to_tomcat	1	1	1	0.46	0.98	0.47	**0.54**	**0.02**	**0.53**
Mylyn_to_velocity-1.6	0.99	1	0.99	0.61	0.8	0.36	**0.38**	**0.2**	**0.63**
Mylyn_to_xalan-2.4	0.85	0.99	0.84	0.53	0.93	0.47	**0.32**	**0.06**	**0.37**
PDE_to_ant-1.7	1	1	1	0.71	0.97	0.64	**0.29**	**0.03**	**0.35**
PDE_to_camel-1.4	0.99	0.99	0.99	0.76	0.98	0.72	**0.23**	**0.01**	**0.27**
PDE_to_jedit-4.0	0.95	1	0.94	0.61	0.88	0.48	**0.34**	**0.12**	**0.46**
PDE_to_tomcat	1	1	1	0.6	1	0.61	**0.4**	0	**0.39**
PDE_to_velocity-1.6	1	1	1	0.65	0.85	0.44	**0.34**	**0.15**	**0.56**
PDE_to_xalan-2.4	0.78	0.99	0.76	0.68	0.97	0.64	**0.1**	**0.02**	**0.12**
tomcat_to_JDT	1	1	1	0.86	1	0.84	**0.14**	0	**0.16**
tomcat_to_Lucene	0.76	1	0.76	0.53	0.99	0.55	**0.23**	**0.01**	**0.21**
tomcat_to_Mylyn	0.83	1	0.82	0.75	1	0.74	**0.08**	0	**0.08**
tomcat_to_PDE	0.72	0.99	0.7	0.74	1	0.72	-0.02	-0.01	-0.02
velocity-1.6_to_JDT	1	1	1	0.78	0.97	0.73	**0.22**	**0.03**	**0.27**
velocity-1.6_to_Lucene	0.77	1	0.77	0.49	0.97	0.5	**0.28**	**0.03**	**0.27**
velocity-1.6_to_Mylyn	0.82	1	0.81	0.67	1	0.65	**0.15**	0	**0.15**
velocity-1.6_to_PDE	0.72	0.99	0.69	0.64	0.99	0.62	**0.07**	-0.01	**0.07**
xalan-2.4_to_JDT	1	1	1	0.86	1	0.84	**0.14**	0	**0.16**
xalan-2.4_to_Lucene	0.76	1	0.76	0.54	0.99	0.55	**0.23**	**0.01**	**0.21**
xalan-2.4_to_Mylyn	0.83	1	0.82	0.75	1	0.74	**0.08**	0	**0.08**
xalan-2.4_to_PDE	0.72	0.99	0.7	0.74	1	0.73	-0.02	-0.01	-0.03
Average	**0.9**	**1**	**0.89**	**0.69**	**0.97**	**0.65**	**0.21**	**0.03**	**0.24**

correlation computations. This suggests that it is likely one of the more important features in the dataset. This hypothesis is further validated in subsequent experiments, as shown in Fig. 8.

Here are the steps for interpretation:

Identify High-Weight Features
1. Locate the darkest areas.
2. Note the feature indices of these areas.

Analyze Feature Relationships
1. Examine the relationships between high-weight features and other features.
2. Identify interactions among high-weight features.

Summarize Observations
1. Use the identified high-weight features and their relationships to optimize the model.
2. Explain the model's predictive behavior based on these observations.

Table 9
Performance on CPDP-IFS model.

CPDP-IFS	Processed			Unprocessed			Improve		
	F_1	AUC	MCC	F_1	AUC	MCC	F_1	AUC	MCC
ant-1.7_to_JDT	1	1	1	0.61	0.78	0.49	**0.39**	**0.22**	**0.51**
ant-1.7_to_Lucene	0.44	0.87	0.46	0.42	0.67	0.28	**0.02**	**0.2**	**0.17**
ant-1.7_to_Mylyn	0	0.22	-0.38	0.48	0.66	0.29	-0.48	-0.43	-0.67
ant-1.7_to_PDE	0.45	0.8	0.41	0.42	0.78	0.39	**0.03**	**0.02**	**0.03**
camel-1.4_to_JDT	1	1	1	0.44	0.61	0.26	**0.56**	**0.39**	**0.74**
camel-1.4_to_Lucene	0.72	0.96	0.72	0.45	0.71	0.34	**0.27**	**0.25**	**0.38**
camel-1.4_to_Mylyn	0.44	0.81	0.42	0.57	0.75	0.43	-0.13	**0.06**	-0.02
camel-1.4_to_PDE	0.94	0.99	0.93	0.43	0.68	0.29	**0.52**	**0.31**	**0.64**
JDT_to_ant-1.7	1	1	1	0.46	0.64	0.26	**0.53**	**0.35**	**0.73**
JDT_to_camel-1.4	0.99	0.99	0.99	0.33	0.69	0.26	**0.66**	**0.31**	**0.73**
JDT_to_jedit-4.0	0	0.01	-0.96	0.47	0.62	0.27	-0.47	-0.61	-1.23
JDT_to_tomcat	0.97	0.99	0.97	0.4	0.68	0.28	**0.57**	**0.31**	**0.69**
JDT_to_velocity-1.6	0.99	0.99	0.99	0.55	0.72	0.42	**0.44**	**0.28**	**0.57**
JDT_to_xalan-2.4	0.68	0.91	0.65	0.4	0.65	0.27	**0.27**	**0.26**	**0.38**
jedit-4.0_to_JDT	0	0	-1	0.42	0.62	0.23	-0.42	-0.61	-1.23
jedit-4.0_to_Lucene	0	0.01	-0.99	0.31	0.61	0.26	-0.3	-0.6	-1.25
jedit-4.0_to_Mylyn	0.69	0.93	0.68	0.36	0.58	0.19	**0.33**	**0.36**	**0.49**
jedit-4.0_to_PDE	0.01	0.15	-0.51	0.36	0.63	0.24	-0.35	-0.48	-0.75
Lucene_to_ant-1.7	1	1	1	0.52	0.73	0.37	**0.47**	**0.27**	**0.62**
Lucene_to_camel-1.4	1	1	1	0.4	0.67	0.26	**0.6**	**0.33**	**0.74**
Lucene_to_jedit-4.0	0	0.09	-0.73	0.5	0.67	0.29	-0.5	-0.58	-1.02
Lucene_to_tomcat	0.96	0.99	0.96	0.31	0.73	0.27	**0.65**	**0.26**	**0.69**
Lucene_to_velocity-1.6	0.99	0.99	0.99	0.56	0.66	0.31	**0.43**	**0.33**	**0.68**
Lucene_to_xalan-2.4	0.56	0.85	0.52	0.37	0.66	0.23	**0.19**	**0.19**	**0.29**
Mylyn_to_ant-1.7	0	0	-1	0.31	0.54	0.06	-0.3	-0.53	-1.05
Mylyn_to_camel-1.4	0	0	-1	0.16	0.52	0.02	-0.16	-0.51	-1.02
Mylyn_to_jedit-4.0	0.99	1	0.99	0.23	0.54	0.06	**0.76**	**0.46**	**0.93**
Mylyn_to_tomcat	0	0.01	-0.96	0.28	0.59	0.13	-0.28	-0.58	-1.09
Mylyn_to_velocity-1.6	0.01	0.01	-0.98	0.55	0.72	0.44	-0.55	-0.71	-1.42
Mylyn_to_xalan-2.4	0.05	0.28	-0.33	0.26	0.62	0.17	-0.21	-0.35	-0.5
PDE_to_ant-1.7	0.96	0.99	0.95	0.33	0.63	0.21	**0.63**	**0.36**	**0.74**
PDE_to_camel-1.4	0.96	0.99	0.95	0.38	0.65	0.26	**0.58**	**0.34**	**0.69**
PDE_to_jedit-4.0	0.97	0.99	0.96	0.57	0.74	0.45	**0.39**	**0.25**	**0.5**
PDE_to_tomcat	0.84	0.98	0.83	0.27	0.63	0.18	**0.57**	**0.35**	**0.66**
PDE_to_velocity-1.6	0.99	0.99	0.99	0.33	0.62	0.21	**0.66**	**0.37**	**0.78**
PDE_to_xalan-2.4	0.79	0.95	0.77	0.35	0.63	0.24	**0.43**	**0.32**	**0.53**
tomcat_to_JDT	1	1	1	0.48	0.66	0.44	**0.52**	**0.34**	**0.56**
tomcat_to_Lucene	0.54	0.91	0.55	0.34	0.64	0.27	**0.2**	**0.27**	**0.28**
tomcat_to_Mylyn	0	0.17	-0.45	0.34	0.62	0.26	-0.34	-0.45	-0.71
tomcat_to_PDE	0.46	0.81	0.42	0.32	0.61	0.21	**0.13**	**0.2**	**0.21**
velocity-1.6_to_JDT	1	1	1	0.48	0.68	0.33	**0.52**	**0.32**	**0.67**
velocity-1.6_to_Lucene	0.45	0.87	0.47	0.29	0.65	0.2	**0.17**	**0.22**	**0.26**
velocity-1.6_to_Mylyn	0	0.17	-0.46	0.31	0.62	0.18	-0.31	-0.45	-0.63
velocity-1.6_to_PDE	0.44	0.79	0.4	0.37	0.66	0.24	**0.08**	**0.13**	**0.16**
xalan-2.4_to_JDT	1	1	1	0.39	0.61	0.18	**0.61**	**0.39**	**0.82**
xalan-2.4_to_Lucene	1	1	1	0.24	0.61	0.14	**0.76**	**0.39**	**0.86**
xalan-2.4_to_Mylyn	1	1	1	0.32	0.63	0.2	**0.68**	**0.37**	**0.8**
xalan-2.4_to_PDE	1	1	1	0.32	0.63	0.19	**0.68**	**0.37**	**0.81**
Average	**0.61**	**0.72**	**0.4**	**0.39**	**0.65**	**0.26**	**0.22**	**0.07**	**0.14**

4.4.2. Experimental group interpretation results

Fig. 8 presents the ranking of metric features based on the total attention weights, where DECISION_DENSITY, HALSTED_CONTENT, and HALSTED_VOLUME emerge as the top three most influential features.

Through visualization, it is evident that strongly correlated features—highlighted by the attention mechanism—positively contribute to the learning process, whereas weakly correlated features may introduce noise. This interpretable mechanism allows for a more intuitive understanding of feature dependencies and the model's internal reasoning patterns.

To assess the effectiveness of the identified features, we conducted comparative experiments using Top-K subsets ranked by attention scores. As presented in Table 13, the baseline model utilizing all available features yielded similar accuracy to the Top-20 and Top-10 subsets, yet demonstrated inferior F1 and MCC scores. Notably, the configuration employing only the Top-1 feature achieved the highest F1 score (0.202) and MCC (0.106), suggesting that this feature alone possesses strong discriminative power in identifying defective instances.

While accuracy showed marginal improvement as more features were included, both F1 and MCC metrics declined, suggesting that additional features might introduce redundancy or noise, ultimately degrading the model's ability to correctly identify the minority

Table 10
Performance on JMST model.

JMST	Processed			Unprocessed			Improved		
	F_1	AUC	MCC	F_1	AUC	MCC	F_1	AUC	MCC
ant-1.7_to_JDT	0.99	1.00	0.99	0.59	0.80	0.47	**0.41**	**0.20**	**0.52**
ant-1.7_to_Lucene	0.39	0.91	0.37	0.33	0.78	0.28	**0.06**	**0.13**	**0.09**
ant-1.7_to_Mylyn	0.31	0.77	0.18	0.35	0.66	0.24	-0.05	**0.11**	-0.06
ant-1.7_to_PDE	0.41	0.84	0.31	0.37	0.73	0.25	**0.04**	**0.11**	**0.06**
camel-1.4_to_JDT	1.00	1.00	0.99	0.53	0.76	0.39	**0.47**	**0.24**	**0.60**
camel-1.4_to_Lucene	0.41	0.92	0.40	0.34	0.80	0.30	**0.06**	**0.12**	**0.10**
camel-1.4_to_Mylyn	0.23	0.65	0.06	0.33	0.67	0.21	-0.10	-0.02	-0.14
camel-1.4_to_PDE	0.36	0.82	0.24	0.37	0.71	0.25	0.00	**0.11**	0.00
JDT_to_ant-1.7	1.00	1.00	1.00	0.52	0.78	0.36	**0.48**	**0.22**	**0.64**
JDT_to_camel-1.4	0.64	0.99	0.60	0.38	0.69	0.23	**0.25**	**0.31**	**0.37**
JDT_to_jedit-4.0	0.82	1.00	0.77	0.48	0.71	0.29	**0.33**	**0.29**	**0.48**
JDT_to_tomcat	0.97	1.00	0.97	0.33	0.79	0.29	**0.64**	**0.21**	**0.67**
JDT_to_velocity-1.6	1.00	1.00	1.00	0.46	0.66	0.19	**0.54**	**0.34**	**0.81**
JDT_to_xalan-2.4	0.28	0.66	0.10	0.42	0.76	0.30	-0.15	-0.10	-0.20
jedit-4.0_to_JDT	0.98	1.00	0.97	0.55	0.79	0.42	**0.42**	**0.21**	**0.55**
jedit-4.0_to_Lucene	0.47	0.98	0.49	0.32	0.79	0.26	**0.15**	**0.19**	**0.23**
jedit-4.0_to_Mylyn	0.35	0.75	0.23	0.32	0.65	0.19	**0.03**	**0.09**	**0.05**
jedit-4.0_to_PDE	0.42	0.80	0.31	0.35	0.69	0.22	**0.07**	**0.12**	**0.09**
Lucene_to_ant-1.7	1.00	1.00	1.00	0.45	0.68	0.24	**0.55**	**0.32**	**0.76**
Lucene_to_camel-1.4	0.81	0.99	0.78	0.41	0.72	0.27	**0.40**	**0.27**	**0.51**
Luceneto_jedit-4.0	0.80	1.00	0.74	0.53	0.78	0.35	**0.27**	**0.22**	**0.40**
Lucene_to_tomcat	0.95	1.00	0.95	0.30	0.76	0.26	**0.65**	**0.24**	**0.69**
Lucene_to_velocity-1.6	1.00	1.00	1.00	0.55	0.73	0.32	**0.45**	**0.27**	**0.68**
Lucene_to_xalan-2.4	0.58	0.76	0.56	0.43	0.78	0.32	**0.15**	-0.02	**0.23**
Mylyn_to_ant-1.7	1.00	1.00	1.00	0.46	0.68	0.26	**0.54**	**0.32**	**0.73**
Mylyn_to_camel-1.4	0.79	0.99	0.77	0.39	0.68	0.24	**0.40**	**0.31**	**0.53**
Mylyn_to_jedit-4.0	0.80	1.00	0.75	0.47	0.70	0.28	**0.33**	**0.30**	**0.47**
Mylyn_to_tomcat	0.55	1.00	0.56	0.29	0.77	0.24	**0.26**	**0.23**	**0.32**
Mylyn_to_velocity-1.6	1.00	1.00	1.00	0.48	0.65	0.17	**0.52**	**0.35**	**0.83**
Mylyn_to_xalan-2.4	0.34	0.60	0.18	0.40	0.73	0.28	-0.07	-0.13	-0.09
PDE_to_ant-1.7	1.00	1.00	1.00	0.54	0.78	0.38	**0.46**	**0.22**	**0.62**
PDE_to_camel-1.4	1.00	0.99	1.00	0.37	0.68	0.21	**0.63**	**0.31**	**0.79**
PDE_to_jedit-4.0	0.81	1.00	0.76	0.44	0.72	0.24	**0.37**	**0.28**	**0.52**
PDE_to_tomcat	0.95	1.00	0.95	0.31	0.77	0.27	**0.64**	**0.23**	**0.68**
PDE_to_velocity-1.6	0.98	1.00	0.97	0.53	0.67	0.26	**0.45**	**0.33**	**0.71**
PDE_to_xalan-2.4	0.46	0.79	0.42	0.43	0.78	0.31	**0.03**	**0.01**	**0.10**
tomcat_to_JDT	0.99	1.00	0.99	0.56	0.79	0.43	**0.43**	**0.21**	**0.56**
tomcat_toLucene	0.53	0.96	0.54	0.29	0.77	0.22	**0.23**	**0.18**	**0.32**
tomcat_to_Mylyn	0.31	0.77	0.18	0.32	0.66	0.19	-0.01	**0.11**	-0.01
tomcat_to_PDE	0.38	0.81	0.28	0.38	0.73	0.26	0.00	**0.08**	**0.01**
velocity-1.6_to_JDT	1.00	1.00	0.99	0.56	0.81	0.43	**0.43**	**0.19**	**0.56**
velocity-1.6_toLucene	0.43	0.90	0.43	0.33	0.78	0.27	**0.10**	**0.12**	**0.16**
velocity-1.6_to_Mylyn	0.29	0.71	0.15	0.32	0.64	0.19	-0.03	**0.07**	-0.04
velocity-1.6_toPDE	0.39	0.80	0.28	0.37	0.72	0.25	**0.02**	**0.08**	**0.03**
xalan-2.4_to_JDT	0.99	1.00	0.99	0.51	0.75	0.37	**0.48**	**0.25**	**0.62**
xalan-2.4_to_Lucene	0.35	0.00	-0.54	0.34	0.76	0.29	**0.01**	-0.76	-0.83
xalan-2.4_to_Mylyn	0.15	0.27	-0.17	0.34	0.66	0.22	-0.19	-0.40	-0.39
xalan-2.4_to_PDE	0.38	0.80	0.27	0.38	0.72	0.26	0.00	**0.08**	0.00
Average	**0.67**	**0.88**	**0.60**	**0.41**	**0.73**	**0.28**	**0.25**	**0.15**	**0.32**

class, which is very important for the task of software defect prediction. This trend underscores the high discriminative power of the attention-based ranking and validates the effectiveness of the Metric Attention Module (MAM) in modeling feature importance.

The experiment also reveals that using only the Top-2 or Top-5 features achieves competitive performance while significantly reducing training time. This balance of interpretability and efficiency makes the MAM a practical choice for software defect prediction tasks.

In conclusion, using the attention weights generated by the Metric Attention Module (MAM) to assess the importance of metric features proves to be an effective strategy. This approach accurately identifies key features and provides an interpretable basis for feature selection. It not only improves training efficiency but also ensures model performance. Therefore, in future work, researchers can flexibly adjust the selection of Top-k features according to specific task requirements to further enhance the model's predictive capability.

To further assess the interpretability and practical utility of feature selection, we compare the classification performance when using only the top-k features ranked by MAM and ReliefF respectively (as shown in Tables 14 and 15). These results offer quantitative insights into how each method contributes to model effectiveness with reduced input dimensionality.

Table 11
Wilcoxon test results: MAM vs. original across CCA+, CLSUP, CPDP-IFS, and JMST.

Method	Metric	MAM (95% CI)	Original (95% CI)	Imp.	p-value
CCA+	F1	0.73[0.71, 0.75]	0.44 [0.42, 0.48]	0.29	<0.05
CCA+	AUC	0.91[0.90, 0.92]	0.75[0.73, 0.77]	0.16	0.078
CCA+	MCC	0.68[0.66, 0.70]	0.32[0.30, 0.36]	0.36	<0.05
CLSUP	F1	0.90[0.88, 0.91]	0.69[0.66, 0.72]	0.21	<0.05
CLSUP	AUC	1.00[0.99, 1.00]	0.97[0.96, 0.98]	0.03	<0.05
CLSUP	MCC	0.89[0.88, 0.91]	0.65[0.61, 0.69]	0.24	<0.05
CPDP-IFS	F1	0.61[0.56, 0.66]	0.39[0.36, 0.42]	0.22	<0.05
CPDP-IFS	AUC	0.72[0.56, 0.66]	0.65[0.63, 0.67]	0.07	<0.05
CPDP-IFS	MCC	0.40[0.35, 0.45]	0.26[0.23, 0.29]	0.14	<0.05
JMST	F1	0.67[0.63, 0.71]	0.41[0.38, 0.44]	0.25	<0.05
JMST	AUC	0.88[0.85, 0.91]	0.73[0.70, 0.76]	0.15	<0.05
JMST	MCC	0.60[0.56, 0.64]	0.28[0.25, 0.31]	0.32	<0.05

Table 12
Performance comparison of raw vs. MAM features under different MLP depths.

Type	Depth	Accuracy	F1	AUC	MCC
Raw	1	0.888	0.105	0.555	0.069
MAM	1	0.908	0.125	0.495	0.163
Raw	2	0.888	0.190	0.500	0.147
MAM	2	0.901	0.118	0.539	0.118
Raw	3	0.888	0.000	0.552	-0.045
MAM	3	0.882	0.100	0.525	0.052

Table 13
Performance with top-k features ranked by attention weights.

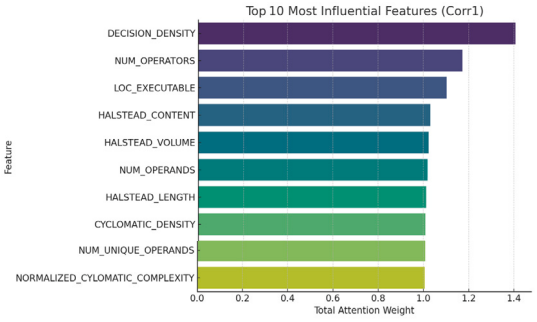
Top-k	Acc.	F1	AUC	MCC	Time (s)
All (baseline)	0.888	0.000	0.718	-0.045	0.062
Top-20	0.888	0.000	0.718	-0.045	0.062
Top-10	0.888	0.000	0.718	-0.045	0.063
Top-5	0.901	0.000	0.750	-0.026	0.060
Top-2	0.836	0.194	0.687	0.104	0.057
Top-1	0.428	0.202	0.576	0.106	0.057

Table 14
Top-10 features ranked by ReliefF score.

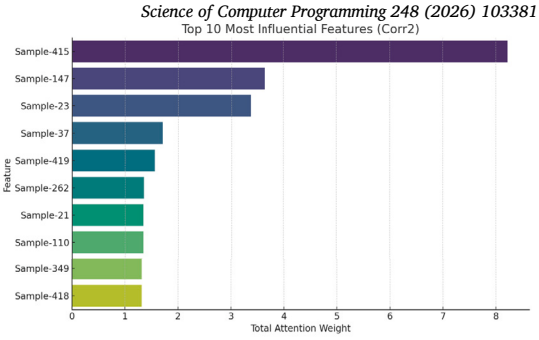
Feature	Score
MAINTENANCE_SEVERITY	0.2299
ESSENTIAL_DENSITY	0.1398
DESIGN_DENSITY	0.1080
NUM_UNIQUE_OPERATORS	0.1044
CALL_PAIRS	0.0997
HALSTEAD_CONTENT	0.0975
NUM_UNIQUE_OPERANDS	0.0900
NUMBER_OF_LINES	0.0876
NUM_OPERATORS	0.0830
HALSTEAD_LENGTH	0.0825

Table 15
Performance with top-k features ranked by ReliefF.

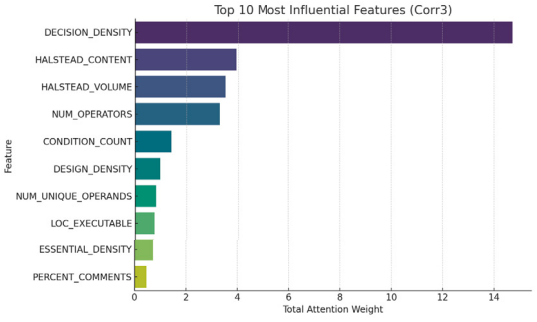
Top-k	Acc.	F1	AUC	MCC	Time (s)
All (baseline)	0.908	0.000	0.724	0.000	0.077
Top-20	0.895	0.000	0.719	-0.037	0.067
Top-10	0.888	0.000	0.738	-0.045	0.062
Top-5	0.888	0.000	0.608	-0.045	0.055
Top-2	0.724	0.125	0.470	-0.007	0.050
Top-1	0.658	0.071	0.399	-0.095	0.050



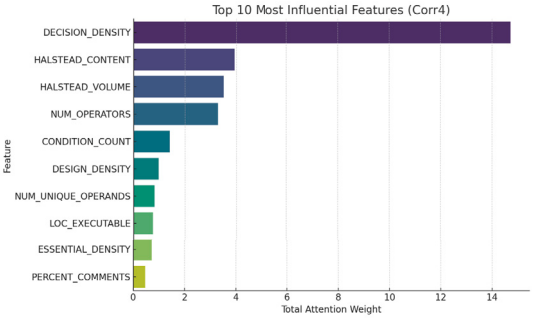
(a) Corr1: Intra-instance feature interaction



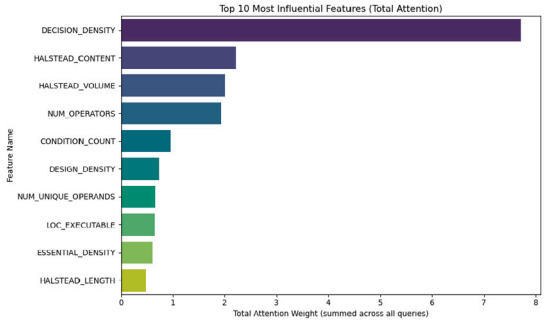
(b) Corr2: Instance-to-instance attention



(c) Corr3: Global feature relationship



(d) Corr4: Cross-instance feature relationship



(e) Total: Weighted Attention Importance

Fig. 8. Top 10 Most Influential Features from Different Attention Views.

Interestingly, while both methods perform similarly when using all features or the top-20/10 ranked ones, MAM exhibits distinct advantages as k becomes smaller. Notably, with only the top-1 feature, MAM yields an F1 score of 0.202 and MCC of 0.106, compared to 0.071 and -0.095 from ReliefF. This suggests that MAM is more capable of identifying highly influential features that are sufficient for minimal yet effective model input. The improvement is particularly evident in metrics that are sensitive to class imbalance, such as MCC and F1.

Furthermore, this contrast supports the claim that MAM provides not only importance ranking but also deeper semantic insights. While ReliefF assigns a single flat score to each feature, MAM aggregates multi-view attentional signals (e.g., Corr1–Corr4), enabling a hierarchical explanation. This is reflected in the differences in the selected top features: although overlapping features like DESIGN_DENSITY and NUM_UNIQUE_OPERATORS appear in both rankings, their prioritization diverges. For instance, MAINTENANCE_SEVERITY is ranked first by ReliefF, while MAM's top attention may highlight a different feature influenced by contextual relationships.

These findings demonstrate that MAM not only improves predictive performance in low-dimensional scenarios but also provides a more nuanced interpretation of feature relevance, making it more suitable for both model compression and explanation-sensitive applications.

5. Discussion

Due to the fact that traditional defect prediction is based on metrics for defect classification, the predictive performance of models depends on the quality of these metrics. However, because the cost of obtaining metric data is relatively high, models perform poorly on certain challenging tasks. Additionally, there are large differences in the feature space of metrics between projects, which means models do not exhibit good generalization ability or the capability for heterogeneous detection. Given the current model and data conditions, finding a reasonable path for improving performance has become the focus of this work.

In this work, the author proposes four types of relationships (see Section 2.5) by mining the metric feature information, and these relationships are effectively integrated through MAM to gain more feature information, enhance model performance, and maintain interpretability. After extensive experimental validation, the proposed technical path and general modules can be generalized in the field of software defect prediction and are not restricted by task environments, models, or other factors.

Based on the above work, the author makes the following discussion.

5.1. Overfitting risk and data limitations

One of the fundamental challenges in software defect prediction (SDP) lies in the scarcity of high-quality, labeled data. Most publicly available datasets are small in scale and often suffer from significant class imbalance. As summarized in $Table2$, the datasets used in this study range from a few hundred to a few thousand instances, with varying degrees of imbalance. This data limitation inherently increases the risk of overfitting and may constrain the generalizability of machine learning models.

In this work, we deliberately chose **not** to apply oversampling (e.g., SMOTE), undersampling, or additional regularization techniques. Our goal was to fairly assess the intrinsic effectiveness of the proposed Metric Attention Module (MAM) across multiple classifiers and datasets without the confounding influence of data-level adjustments. This controlled setup allows us to isolate and better understand MAM's contribution to representation learning and performance gains.

We acknowledge, however, that this choice may also leave the models more exposed to the effects of small sample sizes and label skewness. As a result, while the experimental results are promising, they must be interpreted with an awareness of these inherent limitations. To mitigate variance and improve stability, we employed 30-times repeated holdout validation throughout our experiments.

Generalizing to more diverse and larger-scale real-world projects remains an open challenge in SDP research. Future work will explore integrating MAM with robust resampling strategies, regularization methods, and domain adaptation techniques to further enhance its applicability and resilience to data constraints. We also refer readers to Section 6 for a broader discussion of external validity.

5.2. Designing more general metrics

The authors propose the Metric-Attention Module (MAM), which identifies and integrates four types of relationships between metric features (See $section2.5$), enhancing both model performance and interpretability. Multiple experimental validations confirm MAM's effectiveness across various task environments and model types. Building on this work, the authors discuss the following key points.

5.3. Designing more universal metrics

Universal metrics are essential for improving a model's adaptability and generalization across diverse datasets and application scenarios. MAM's interpretability (See $section4.4$) indicates that not all metric features are effective, highlighting the need for more universal metrics in future research. To achieve this, the authors propose establishing unified metric standards to ensure consistency across different domains, reducing the complexity of feature selection and preprocessing. These metrics should be applicable beyond software defect prediction to fields like medical data analysis and financial forecasting, validating their broad applicability and stability. Flexible metric adaptation is also recommended, involving scalable metrics that adjust based on data characteristics and model needs. Combining multiple metrics and employing data-driven optimization techniques can further enhance predictive performance. Finally, establishing a universal metrics repository will encourage collaboration and continuous improvement in metric design and optimization. Future research should focus on validating these metrics and advancing the associated technologies.

5.4. Enhancing model explainability

Model explainability is crucial in software engineering, especially for tasks like bug fixing, where understanding the decision-making process is essential. While some debate the necessity of explainability in AI, it remains a key standard for evaluating defect prediction models. Explainability methods can be categorized into pre-modeling, inherently explainable model construction, and post-modeling techniques. In this work, MAM's explainable features were used to analyze metric feature roles in decision-making (see $section4.4$), a post-modeling method. However, few methods currently exist for software testing model explainability. Future research should focus on enhancing the inherent explainability of these models and developing comprehensive explanatory tools and workflows.

5.5. Multi-domain extension—data augmentation

Data scarcity is a significant challenge in software prediction due to data acquisition difficulties and privacy concerns. Extracting more information from limited data is crucial. While data augmentation techniques are well-developed in fields like natural language processing and computer vision, they are less common in software defect data. MAM effectively filters important features, offering an approach that could be extended across multiple domains. Future work will focus on leveraging MAM's explainability to perform targeted metric data enhancements, addressing data scarcity and improving the effectiveness of data augmentation techniques in software prediction.

6. Threats to validity

This study identifies threats to validity in construct, internal, and external categories, crucial for ensuring robust and generalizable findings.

Construct validity examines whether the study measures the intended outcomes accurately. While the Metric Attention Module (MAM) aims to improve defect prediction models' interpretability and performance, parameter sensitivity poses a threat, as varying hyperparameter settings could affect MAM's perceived benefits. Metric selection and transformation by MAM may introduce biases, impacting feature importance and interpretability measures, which, despite validation through visualization, might still suffer from subjective interpretation and risks of overfitting.

Internal validity concerns whether the results are genuinely due to the tested interventions. Limited explanation techniques could constrain understanding of MAM's effectiveness compared to other methods, suggesting a need for broader comparisons. Variations in model architectures, such as neural network complexity, could also affect MAM's impact, necessitating careful attribution of performance gains.

External validity relates to the generalizability of findings. The study's reliance on datasets like AEEEM, JIRA, PROMISE, and NASA may not represent all real-world scenarios, impacting result applicability. Task environment variations, such as defect types and project sizes, further challenge generalizability. The limited number of tested models and environments may also restrict broader applicability, highlighting the need for future research to explore more models, tasks, and datasets.

Addressing these threats is essential for understanding MAM's impact on defect prediction, and future studies should include diverse datasets, explanation techniques, and comparative analyses.

7. Conclusion

This study introduced the Metric Attention Module (MAM) to enhance software defect prediction (SDP) by using attention mechanisms to explore complex relationships within and between metric data features. MAM significantly improves model performance in within-project, cross-project, and heterogeneous defect prediction tasks, as demonstrated on standard datasets.

Results show that MAM enhances interpretability and performance by effectively capturing inter-metric and inter-instance relationships, achieving average improvements of 40% in F1 score, 20% in accuracy, and 50% in MCC over baseline models, indicating its robustness and adaptability.

Despite these advances, performance varied across task environments, suggesting a need for further refinement for broader applicability. Future work could integrate MAM with other techniques to address these challenges and extend its use in other software engineering areas.

Overall, this research highlights the potential of attention mechanisms in SDP, offering a new approach to improving both accuracy and interpretability of defect prediction models.

CRediT authorship contribution statement

Yongchang Ding: Writing – review & editing, Writing – original draft, Software, Resources, Project administration, Methodology, Formal analysis, Data curation. **Wei Han:** Validation. **Zhiqiang Li:** Validation. **Haowen Chen:** Writing – review & editing, Validation. **Linjun Chen:** Writing – review & editing, Software. **Rong Peng:** Writing – review & editing, Writing – original draft, Supervision, Resources, Project administration, Conceptualization. **Xiao-Yuan Jing:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Investigation, Funding acquisition, Formal analysis.

Funding

Project Acknowledgements: This work was supported by the NSFC Project No. 62176069, the NSF of Guangdong Province No. 2023A1515012653, the Innovation Group of Guangdong Education Department under Grant No. 2020KCXTD014, and the 2019 Key Discipline project of Guangdong Province.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Implementation details and reproducibility

To ensure reproducibility of our proposed Metric Attention Module (MAM), we provide a detailed summary of our experimental pipeline and implementation. The full source code and datasets used in our experiments will be made publicly available at: https://github.com/HUICHUANVISION/METRRIC_ATTENTION_MODULE.git.

A.1. Experimental environment

- Language: Python 3.10
- Frameworks: PyTorch 2.0.1, scikit-learn 1.2.2
- Libraries: NumPy, pandas, seaborn, matplotlib
- Hardware: Intel i7 CPU, 32GB RAM, no GPU required

A.2. Model architecture

The MAM module is implemented as a PyTorch neural network module that processes standardized tabular metric data. The attention mechanism follows the scaled dot-product formulation with learnable projection layers for query (Q), key (K), and value (V). Four branches extract different types of relational information:

- **Corr1:** Inter-metric feature relationships within instance
- **Corr2:** Inter-instance feature relationships
- **Corr3:** Correlation within metric features
- **Corr4:** Correlation between single metric features

The outputs of all four branches are dynamically weighted using parameters $\alpha, \beta, \gamma, \delta$ (default 0.25), fused, and multiplied element-wise with the original feature matrix to yield enhanced representations.

A.3. Training and evaluation protocol

- Datasets Used:
  - AEEEM: EQ, JDT, Lucene, Mylyn, PDE
  - NASA: CM1, MW1, PC1, PC3, PC4
  - JIRA: activemq-5.0.0, derby-10.5.1.1, groovy-1_6_BETA_1, hbase-0.94.0, hive-0.9.0, jruby-1.1, wicket-1.3.0-beta2
  - PROMISE: ant-1.7, camel-1.4, ivy-2.0, jedit-4.0, log4j-1.0, poi-2.0, tomcat, velocity-1.6, xalan-2.4, xerces-1.3
- Preprocessing: StandardScaler normalization (mean 0, variance 1)
- Classifier Settings:
  - Random Forest (RF): `n_estimators=100`, `class_weight='balanced'`, `random_state=42`
  - Logistic Regression (LR): `solver='liblinear'`, `penalty='l2'`, `class_weight='balanced'`, `random_state=42`
  - Decision Tree (DT): `criterion='gini'`, `class_weight='balanced'`, `random_state=42`
- Evaluation Metrics: Accuracy, F1-Score, AUC, MCC
- Train/Test Split: 70%/30%, stratified on label distribution

References

[1] G. O'Regan, Ethical and Legal Aspects of Computing - A Professional Perspective from Software Engineering, Undergraduate Topics in Computer Science, Springer, 2024, Available online.
[2] S. Chenoweth, P.K. Linos, Teaching machine learning as part of agile software engineering, IEEE Trans. Ed. 67 (3) (2024) 377–386, https://doi.org/10.1109/TE.2023.3337343.
[3] N.E. Fenton, M. Neil, A critique of software defect prediction models, IEEE Trans. Softw. Eng. 25 (5) (1999) 675–689.
[4] T. Gyimothy, R. Ferenc, I. Siket, Empirical validation of object-oriented metrics on open source software for fault prediction, IEEE Trans. Softw. Eng. 31 (10) (2005) 897–910.
[5] P. Shaw, J. Uszkoreit, A. Vaswani, Self-attention with relative position representations, arXiv preprint, arXiv:1803.02155, 2018, Available online: https://arxiv.org/abs/1803.02155.
[6] X. Yang, D. Lo, X. Xia, et al., Deep learning for just-in-time defect prediction, in: Proceedings of the 2015 IEEE International Conference on Quality, Reliability, and Security (QRS), 2015, Available online.
[7] J. Li, P. He, J. Zhu, M.R. Lyu, Software defect prediction via convolutional neural network, in: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 318–328.
[8] F. Matloob, T.M. Ghazal, N. Taleb, S. Aftab, M. Ahmad, M.A. Khan, S. Abbas, T.R. Soomro, Software defect prediction using ensemble learning: a systematic literature review, IEEE Access 9 (2021) 98754–98771.
[9] R.E. Al-Qutaish, A. Abran, An analysis of the design and definitions of Halstead's metrics, in: Proceedings of the 15th International Workshop on Software Measurement (IWSM'05), 2005.
[10] T.J. McCabe, A complexity measure, IEEE Trans. Softw. Eng. SE-2 (4) (1976) 308–320.
[11] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Trans. Softw. Eng. 20 (6) (1994) 476–493.
[12] F. Abreu, MOOD-metrics for object-oriented design, in: Addendum to Proc. OOPSLA, vol. 94, 1994.

[13] P. Bhattacharya, M. Iliofotou, I. Neamtiu, et al., Graph-based analysis and prediction for software evolution, in: Proceedings of the 2012 34th International Conference on Software Engineering (ICSE), IEEE, 2012, pp. 419–429.
[14] T. Lee, J. Nam, D.G. Han, et al., Micro interaction metrics for defect prediction, in: Proceedings of the 19th ACM SIGSOFT Symposium and 13th European Conference on Foundations of Software Engineering, 2011, pp. 311–321.
[15] T. Jiang, L. Tan, S. Kim, Personalized defect prediction, in: Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2013, pp. 279–289.
[16] A.S. Nuñez-Varela, H.G. Pérez-Gonzalez, F.E. Martínez-Perez, C. Soubervielle-Montalvo, Source code metrics: a systematic mapping study, J. Syst. Softw. 128 (2017) 164–197, https://doi.org/10.1016/j.jss.2017.03.044, Available online.
[17] L. Madeyski, M. Jureczko, Which process metrics can significantly improve defect prediction models? An empirical study, Softw. Qual. J. 23 (2015) 393–422, https://doi.org/10.1007/s11219-014-9241-7, Available online.
[18] P. He, B. Li, X. Liu, J. Chen, Y. Ma, An empirical study on software defect prediction with a simplified metric set, Inf. Softw. Technol. 59 (2015) 170–190, https://doi.org/10.1016/j.infsof.2014.11.006, Available online.
[19] P. Chlap, H. Min, N. Vandenberg, J. Dowling, L. Holloway, A. Haworth, A review of medical image data augmentation techniques for deep learning applications, J. Med. Imag. Radiat. Oncol. 65 (5) (2021) 545–563, https://doi.org/10.1111/1754-9485.13261, Available online.
[20] W.A. Kamakura, M. Wedel, F. De Rosa, J.A. Mazzon, Cross-selling through database marketing: a mixed data factor analyzer for data augmentation and prediction, Int. J. Res. Mark. 20 (1) (2003) 45–65, https://doi.org/10.1016/S0167-8116(02)00120-3, Available online.
[21] C. Shorten, T.M. Khoshgoftaar, B. Furht, Text data augmentation for deep learning, J. Big Data 8 (1) (2021) 101, https://doi.org/10.1186/s40537-021-00492-0, Available online.
[22] C. Shorten, T.M. Khoshgoftaar, A survey on image data augmentation for deep learning, J. Big Data 6 (1) (2019) 1–48, https://doi.org/10.1186/s40537-019-0197-0, Available online.
[23] Y. Ding, et al., A supervised data augmentation strategy based on random combinations of key features, Inf. Sci. 632 (2023) 678–697.
[24] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (1) (2006) 2–13.
[25] J. Nam, S. Kim, Clami: defect prediction on unlabeled datasets, in: Proc. 30th Int. Conf. Automated Software Eng. (ASE), 2015, pp. 1–12.
[26] Q. Yu, S. Jiang, Y. Zhang, A feature matching and transfer approach for cross-company defect prediction, J. Syst. Softw. 132 (2017) 366–378.
[27] A. Maćkiewicz, W. Ratajczak, Principal components analysis (PCA), Comput. Geosci. 19 (3) (1993) 303–342.
[28] P. Xanthopoulos, P.M. Pardalos, T.B. Trafalis, Linear discriminant analysis, in: Robust Data Mining, 2013, pp. 27–33.
[29] J.D. Carroll, P. Arabie, Multidimensional scaling, in: Measurement, Judgment and Decision Making, 1998, pp. 179–250.
[30] S. Pal, A. Sillitti, Cross-project defect prediction: a literature review, IEEE Access 10 (2022) 118697–118717.
[31] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empir. Softw. Eng. 17 (4–5) (2012) 531–577.
[32] T. Diamantopoulos, A.L. Symeonidis, Mining Software Engineering Data for Software Reuse, Springer, 2020.
[33] T. Zimmermann, N. Nagappan, Predicting defects using network analysis on dependency graphs, in: 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, 2008, pp. 531–540.
[34] H. Chen, X.-Y. Jing, B. Xu, Heterogeneous defect prediction through joint metric selection and matching, in: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), 2021, pp. 367–377.
[35] X.-Y. Jing, F. Wu, X. Dong, F. Qi, B. Xu, Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015), Bergamo, Italy, 2015, pp. 496–507.
[36] Z. Li, X.-Y. Jing, X. Zhu, Heterogeneous fault prediction with cost-sensitive domain adaptation, Softw. Test. Verif. Reliab. 28 (2) (2018).
[37] P. He, B. Li, Y. Ma, Towards cross-project defect prediction with imbalanced feature sets, CoRR, arXiv:1411.4228, 2014.
[38] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. 34 (4) (2008) 485–496.
[39] N. Spolaôr, E.A. Cherman, M.C. Monard, H.D. Lee, ReliefF for multi-label feature selection, in: Proc. Brazilian Conf. Intell. Syst. (BRACIS), 2013, pp. 6–11.
[40] F. Wilcoxon, Individual comparisons by ranking methods, in: Breakthroughs in Statistics: Methodology and Distribution, Springer, New York, NY, USA, 1992, pp. 196–202.
[41] J. Nam, S.J. Pan, S. Kim, Transfer defect learning, in: Proc. 35th Int. Conf. Software Eng. (ICSE), 2013, pp. 382–391.
[42] X. Jing, F. Wu, X. Dong, F. Qi, B. Xu, Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning, in: Proc. 2015 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 496–507.
[43] Z. Li, X.-Y. Jing, F. Wu, X. Zhu, B. Xu, S. Ying, Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction, Autom. Softw. Eng. 25 (2018) 201–245.