تجربة التنبؤ بكثافة عيوب البرمجيات باستخدام نقاط الانتباه للبيانات والتعلم العميق

د. إيمان "محمد نعيم" ياسين*

تاريخ استلام البحث: 2024/4/11م تاريخ قبول البحث: 2024/8/7م

الملخص

هذه الدراسة تهدف إلى مواجهة مشكلة توقع كثافة عيوب البرمجيات من خلال استخدام ما تقدم به العلم من تقنيات التعلم الآلي والتعلم العميق. سيتم هذا من خلال دمج آليات الانتباه، مثل الانتباه الذاتي والانتباه المتقاطع، نقوم بتقديم نموذجاً يُسمىAttDeep

هذا النموذج صمم لتحسين دقة توقع عيوب البرمجيات. تُستخدم آليات الانتباه على التركيز على السياق الموجود في البيانات مما يعزز قدرته على اكتشاف الأنماط التي قد تشير إلى عيوب محتملة. من خلال سلسلة من التجارب باستخدام مجموعات بيانات من داتا سيت معروفة، تظهر نتائجنا أن نموذج AttDeep يؤدي أداءً أفضل واخطاء اقل وافاد ذكلك بإظهار تحسناً في التعميم مقارنةً بالنماذج التي لا تستخدم آليات الانتباه. تسلط هذه النتائج الضوء على فعالية دمج آليات الانتباه مع التعلم لتوقع عيوب البرمجيات مما يساهم في مسار للبحوث المستقبلية والتطبيقات العملية، في ضمان جودة البرمجيات

الكلمات المفتاحية: التعلم العميق؛ كثافة العيوب؛ التنبؤ بعيوب البرمجيات؛ آلية الانتباه.

* أستاذ مساعد، قسم نظم المعلومات الإدارية، جامعة العلوم الإسلامية العالمية.

Experimenting Software Defect Density Prediction using Data Attention Point and Deep Learning

Abstract

Predicting software defect density is crucial for developing reliable software. This research leverages advancements in machine learning and deep learning techniques to enhance the prediction of software defect density. By integrating attention mechanisms, like self-attention and cross attention into a deep learning framework we introduce a model named AttDeep. This model aims to enhance the accuracy of software defect prediction. The attention mechanisms help in reflecting the content across all data parts thereby improving its capability to identify patterns and anomalies that could signal defects. Through experiments using datasets from several repositories our findings indicate that AttDeep not only delivers performance with reduced loss and mean absolute error (MAE) but also exhibits enhanced generalization compared to models without attention mechanisms. These results highlight the effectiveness of incorporating attention mechanisms into deep learning for software defect prediction suggesting avenues, for research and practical applications by integrating attention mechanisms into machine learning algorithms.

Keywords: Deep learning; Defect Density; Software Defect Prediction; Attention Mechanism

Introduction:

Detecting and addressing bugs in the early stages of development not only enhances the overall quality of products but also minimizes the operational, social, and financial challenges associated with software defects [1-3]. Typically fixing bugs found during maintenance costs a huge amount of money, around fourteen thousand and hundred two ($14,102) on average [4]. As a result, Software Defect Prediction (SDP) has emerged as a proactive strategy to prevent potential defects from impacting end users. Numerous methods, for defect prediction, have been developed to detect issues and reduce costs associated with software defects [5–9]. SDP involves the use of different ML techniques to predict the potential defects in software. It's an important step to identify potential issues early in the development process, allowing developers to focus on high-risk areas and mitigate defects before they affect the final product [44].

These methods utilize software data to create models, which are then used to predict the potential of new occurrences of defects.

Software Defect Prediction utilizes machine learning (ML) and Artificial Intelligence (AI) techniques to predict and detect software flaws, in the development process [6], [10]–[12]. By examining information and patterns, SDP models can identify risk areas enabling developers to address issues proactively before they escalate. This proactive approach enhances the quality of software. Reduces the likelihood of errors, during deployment and maintenance phases. Deep learning, a subset of ML has attracted attention for its ability to recognize patterns in data independently. Essentially, deep learning relies on neural networks (NN) with layers that allow the model to construct data representations. These NN mimic the brain's operations with each layer processing information before passing it on for refinement [13].

Deep learning presents an adaptable method for tackling challenges across various domains [14–17]. Its aptitude for assimilating datasets autonomously extracting

features and achieving cutting-edge performance in tasks positions. It is a revolutionary technology with profound implications for the future of AI and data-driven decision-making. Attention mechanisms have gained attraction in the ML community recently due to their ability to enhance model performance across various tasks [18]. These mechanisms provide insights into complex NN models, making them valuable for understanding how models make decisions [19]. The introduction of the Transformer model represented a significant advancement in attention research by showcasing how attention can achieve cutting-edge results [18-20]. Originally intended for tasks like machine translation, this model highlighted how attention can capture intricate relationships within sequential data [2]. Our study focuses on incorporating attention mechanisms into SDP. While most existing research delves into sequence modelling and feature extraction using convolutional neural networks (CNN) [21–23] and recurrent neural networks (RNN) [23-24], there is a lack of exploration in integrating attention mechanisms with handcrafted features.

In this paper, we introduce a new method for SDP by utilizing attention mechanisms alongside handcrafted features. As far as we know, this study marks the initial effort to combine attention mechanisms with manually selected features in the realm of SDP.

Building on prior research, our goal is to showcase how attention mechanisms can effectively capture deep patterns for predicting software defect datasets. This task is essential for predicting software issues as it seeks to illustrate how attention mechanisms can impact every aspect of the dataset ultimately improving the precision of defect prediction.

Through the fusion of attention mechanisms with deep learning structures, we strive to create a robust model that can predict software defects accurately while

ensuring interpretability and performance. We introduce our groundbreaking model (AttDeep) that combines attention mechanisms with feedforward deep learning structures that bring advantages to ML field. This paper is structured as follows: In the next section, we will discuss the related work and existing research. Section 3 will introduce the attention mechanism used while Section 4 will elaborate on the model put forth including the dataset and experimental configuration. The results and discussion will be presented in Section 5. Section 6 will highlight the study's contributions. Lastly, Section 7 will offer the conclusion.

Related Work:

Software Defect Prediction Using Deep Learning:

There has been extensive research on how deep learning methods can be used to predict software defects, supporting high-quality assurance in software. Yet, when we zoom in on predicting the defect density of bugs, we find it clear that this particular area hasn't been thoroughly investigated. This finding suggests a path for studies where the power of deep learning could be utilized to forecast not only the existence of defects but also their density providing a deeper insight into software quality. Measuring defect density is a crucial metric in fields like software development, materials science, and semiconductor manufacturing as it provides information on quality and dependability. The concept of Defect Density, in the field of software metrics in the context of improving software quality and efficiency has been extensively discussed in numerous important texts and academic discussions since the 1970s [25-28]. Limited research exists on the prediction of software defect density, with only a single study, to my knowledge, exploring the topic through deep learning [29-42]. In their research paper [29] pointed out a challenge in predicting defect density due to sparsity issues which could affect the accuracy of prediction. To address this issue, researchers have explored the use of deep learning techniques known for their ability to handle datasets effectively.

The study compared learning with other ML methods using 28 public datasets and found that deep learning excelled in scenarios with high levels of data sparsity and also performed well in situations with moderate or low sparsity rates. This study highlights the potential of deep learning as a tool for predicting defect density, which can greatly improve software quality and reliability. Only a few research papers have tried to predict defect density through a range of ML methods other than deep learning. In a study for [12] a new technique was proposed that uses predictor variables based on defect acceleration measures such as density defect velocity and defect introduction time. The study thoroughly examines how these variables are connected to the number of defects. The study makes use of an integrated ML approach utilizing regression models created from the predictor variables mentioned earlier. This choice of methodology is crucial as it aligns with the increasing trend of using ML techniques to improve accuracy in SDP. An extensive experiment was carried out using ten datasets from the PROMISE repository totaling 22,838 instances, which forms a basis for the study's conclusions. Significantly, the regression model that operates based on defect velocity achieved an adjusted R value of 98.6% with a highly significant p-value below 0.001. This signifies a positive relationship between average defect velocity and the number of defects with a correlation coefficient of 0.98. The high level of predictability indicates the potential for this ML-driven approach to be a guide, for software testing practices. Another study [30] introduced a "fuzzy logic-based approach for phase-wise software defects prediction". This approach is validated using data from 20 software projects includes a detailed sensitivity analysis of the software metrics. The findings from this analysis are valuable for assessing the seriousness of defects across stages of the software development life cycle (SDLC). Relying solely on detecting software defects at the end of testing is often seen as advantageous as making necessary changes in SDLC stages

could lead to significant costs and efforts to achieve desired software quality. Therefore, having a model that predicts defect density indicators at each phase is crucial.

This mentioned study aims to present a logic-based model for predicting software defects in phases by utilizing key reliability-related metrics specific to each stage. Their proposed model predicts defect density indicators during SDLC by considering nine software metrics from these phases. The indicated defect density, projected after each stage is also viewed as a factor influencing the following phase. Analyzing software metrics in context and utilizing an inference system play crucial roles in shaping this model.

The findings confirm the reliability of the suggested model by analyzing data from twenty software projects with the validation results proving satisfactory. Metrics, like the error and the balanced average relative error notably decrease as the scale of the software project expands. Furthermore, a comprehensive survey on the application of deep learning for software defect prediction has been provided, highlighting various methodologies and their effectiveness [43].

Table (1) Related work :

Category	Subcategory	Details	References
Deep Learning in SDP	General Research	Deep learning methods are being explored to predict software defects for high-quality assurance.	[25-28], [29-42]
	Defect Density Prediction	Limited studies on predicting defect density using deep learning; issues with data sparsity.	[29-42]
	Comparison with ML Methods	Deep learning excels in scenarios with high data sparsity; effective compared to other ML methods.	[12], [30]

Category	Subcategory	Details	References
Machine Learning in SDP	Integrated ML Approach	Use of regression models with predictor variables like defect velocity; high correlation and predictability.	[12]
	Fuzzy Logic-Based Approach	Phase-wise defect prediction using fuzzy logic; validated with data from 20 projects.	[30]
Attention Mechanisms in ML	Importance and Applications	Attention mechanisms enhance performance, interpretability, used in various ML applications.	[18], [31-35]
	Transformer Model	Origin of attention mechanisms; widely used beyond NLP, including SDP.	[18], [19], [34-36]
Attention Mechanisms in SDP	BSLDP System	Combines bidirectional LSTM and self-attention for defect prediction; improved F1 scores significantly.	[36]
	Transformer Models	Uses syntax trees and self-attention for defect prediction; outperforms CNN models.	[37]
	CNN-GSA Model	Combines CNN with various attention mechanisms for defect prediction; superior F1 measure performance.	[38]
	MobileNetV2 with Attention	Lightweight model with MobileNetV2 and attention for object classification; outperforms other deep learning.	[39]

Category	Subcategory	Details	References
	DeepAtt Model	Focuses on attention mechanism in machine translation; enhances translation accuracy.	[40]
	Attention and ML Techniques	Combines learning and attention for defect prediction at method level; significant improvements in F1 and AUC	[41]

Self-Attention and Cross-Attention:

Attention mechanisms have become increasingly important in the field of ML for several reasons. Firstly, models that utilize attention have been able to Obtain exceptional performance. Furthermore, attention mechanisms can be trained together with base models like NN or CNN using standard backpropagation techniques [31]. In addition, attention brings an element of interpretability to NN models which are generally complex and hard to interpret [32]. The rise in popularity of attention mechanisms can be attributed to the success of the transformer model [18], which highlighted the power of attention in achieving results. Originally, the attention mechanism was conceived as an enhancement to RNN used in the transformer model [33]. The mechanism represents an advancement in attention research by demonstrating that attention alone can create cutting-edge models. Like the attention mechanism, the transformer model was initially developed for machine translation but quickly found utility in various applications including image processing [19], video processing [34] recommender systems [35] and software defect prediction [36]. Our research paper focuses on how attention mechanisms are used in software defect prediction. Most of the research papers tackle issues related to meaning and sequence along, with extracting features using techniques like CNN and RNN. Interestingly,

attention mechanisms have not been combined with hand-crafted features, in this scenario before.

Attention mechanisms have been commonly used alongside several ML in different contexts. Their utilization in software defect prediction has not been extensively explored.

Although some papers have delved into combining attention mechanisms with deep learning approaches, only a few have focused on their effectiveness in the field of SDP. For example, in a study [36], researchers highlighted the importance of considering data distribution variations for software defects in cross-projects, particularly focusing on semantic depth. The researchers created a system called BSLDP to improve the precision SDP in projects. This system uses a model that merges bidirectional LSTM networks with self- attention mechanisms. This special combination is intended to examine source code files effectively exploring for meanings and pattern that could signal possible defects. One key component they introduced is ALC, an extractor that captures semantics from source code and feeds it into the BSL classification algorithm for building the predictive model. Additionally, they proposed a meshing mechanism where ALC breaks down numerical token vectors to generate details on smaller segments thereby boosting the model's performance.

The model's effectiveness was evaluated using the available PROMISE dataset and compared against four cutting-edge methods. The results showed that BSLDP improved the project defect prediction performance in terms of F1 by an average of 14.2%, 34.6%, 32.2% and 23.6% respectively. Another study [37] presents a technique for anticipating software issues through the use of Transformer models, which depend on self-attention mechanisms to grasp grammar and meaning in software systems. The suggested method involves transforming software components

into syntax trees extracting word sequences and using self-attention layers to embed features. Lastly, a Softmax is utilized for defect forecasting.

Experimental findings indicate that the Transformer approach surpasses CNN models by an average of 3.2% across three open-source software projects demonstrating its effectiveness in predicting software defects. Moreover, extensive research has combined attention mechanisms with learning in a range of areas. For example, researchers have investigated how attention mechanisms can be used in tasks, like machine translation and sentiment analysis within natural language processing (NLP). Furthermore, attention mechanisms have also been utilized in computer vision tasks such as recognizing objects within images and identifying items in photos and creating captions for images to focus on areas of input data. These studies highlight the adaptability and efficiency of attention mechanisms in improving the effectiveness of learning models across fields and applications. One research paper [38] introduces an approach for predicting defects called CNN-GSA, which overcomes the drawbacks of existing visualization methods by integrating self-attention mechanisms. The approach transforms code into images Utilizing CNN with "channel attention, spatial attention and self- attention mechanisms to extract structural and semantic aspects" from these code images. The study's results show that the CNN-GSA model outperforms techniques in prediction tasks based on F1 measure results underscoring its ability to capture comprehensive information crucial for defect prediction in software quality assurance. Some authors applied the attention mechanism with CNN deep learning such as Shahiet al [39]. The authors of the study put forward a learning model that's lightweight and incorporates the pre-trained MobileNetV2 model along with the attention mechanism. They describe their approach as involving two steps; first extracting features to capture important information about objects and second using an attention module

to focus on the relevant semantic details. By combining these components with adding deep learning layers and a softmax layer they.

create a comprehensive model. To validate their method the authors evaluate it using three datasets related to fruits. Their results show that their proposed approach, which utilizes transfer learning outperforms four deep learning methods. Importantly, their model achieves performance while requiring trainable parameters and maintaining high accuracy in classification tasks. In their study, Zhang, Xiong and Su [40] delve into the importance of the attention mechanism, specifically in the context of machine translation tasks.

While previous research focused mainly on enhancing the encoder and decoder architectures for machine translation, little attention was given to the attention mechanism itself.

They highlight that the attention mechanism plays a role in establishing translation correspondence between languages, especially when shallow neural networks may struggle to capture intricate relationships effectively. To bridge this gap, the authors propose an attention model called DeepAtt. This model utilizes attention information to determine which information should be highlighted or deleted within encoder layers. Doing this process ensures the creation of distributed representations that are conducive to high-level attention and accurate translation. Through their experiments, they.

demonstrate that their DeepAtt model, with five attention layers, achieves great performance compared to other results.

Furthermore, they observe empirically that increasing the number of attention layers enhances the accuracy of attention weights thus improving translation fidelity. Moreover, the authors demonstrate how DeepAtt greatly improves the accuracy and reliability of system-generated translations. This emphasizes the significance of

attention mechanisms in machine translation models and brings attention to the potential for future progress in their field. Another research paper [41] presents an approach that combines learning techniques and attention mechanisms. The main goal of the paper is to provide developers with prediction information by identifying potentially problematic lines of code when predicting defects at the method level. The proposed model utilizes

syntax trees as a representation of code snippets. To handle the nature of defect data, the model incorporates a Self-organizing Map clustering approach to deal with noisy data.

Experimental evaluations show improvements achieved by the proposed model. On average, it improves the F1 measure by 17.7% and the AUC by 37.8% when compared to four ML algorithms. This demonstrates the effectiveness and potential of using learning and attention-based approaches to address complexities in SDP providing developers with insights into potential defects at a more detailed level. As far as we know ,our research paper is the first to tackle using an attention mechanism with hand-crafted features in the software defect prediction field. Most scholars have mainly focused on using the attention mechanism to capture sequence patterns and semantic subtleties. However, our paper diverges from this trend. Our emphasis lies in utilizing the mechanism to mirror the significance and information contained in each element across all data samples and features. This detailed approach enables an examination of the meanings and revelations within the dataset providing a distinctive viewpoint, in the domain of attention-based approaches.

Attention Mechanism:

The method we're employing to transform data for handcrafted features is similar to the attention mechanism presented in the transformer model albeit with some modifications and alterations to fit the particular requirements of handcrafted data. Even though the core concepts of attention like calculating dot products and

normalising softmax,stay aligned to the initial transformer model, how attention weights are applied and understood is customized to suit the features and needs of the manually crafted data utilized in our study setting. The attention system consists of two parts; self-attention and cross-attention [18]. Self-attention in our proposed model is used to grasp the knowledge, in the training data and distribute it among all training instances. It allows each element in the sequence to interact with others aiding in modeling connections and understanding relationships intrinsic to the data. This feature proves beneficial when comprehending the organization and interconnections within the training data holds significance. On the other hand, cross attention takes this idea further by enabling elements in the training dataset to reflect on elements in the testing dataset. This helps in transferring insights from both datasets allowing the model to better understand connections and relationships among data sources. Cross-attention is crucial for harmonizing information across datasets or domains ensuring that the model can adapt effectively to unseen testing data while utilizing the expertise gained during training.

3.1 Self-Attention Applied to The Training Dataset:

In the Transformer model, self-attention is utilized to understand the relationships between input tokens in a sequence.

Each token looks at all tokens in the sequence to create a summary of the entire sequence [18]. Likewise in this custom-built data context, self-attention is calculated among the vectors that represent samples in the training dataset. Each sample pays attention to all samples to determine a weight that reflects its importance within the dataset.

The dot product operation, between vectors of two samples, mirrors the product used in the attention mechanism capturing how similar or correlated these samples are. To ensure accuracy, the softmax function is employed on these weights to

generate normalized attention scores. This guarantees that all weights add up to one and form a probability distribution. Finally, the characteristics of each sample are enhanced by multiplying them with their softmax weights. This enables the model to prioritize features while reducing the impact of significant ones. Self-attention is performed as follows: Let Trn be the training dataset with x dimension. Samples in the TrnX (Target excluded) are represented by vectors Trn_1, Trn_2, . . . , Trn_x, where Trn_i is the i-th vector. The expression for $weight_i$ is given by:

$$\text{weight}_i = \sum_{i,j=1}^{x} \text{Trn}_i \text{Trn}_j$$

Softmax for each $weight_i$ is then calculated using the following equation:

$$\text{Softmax}(\text{weight}_i) = e^{\text{weight}_i} / \sum_{k=1}^{x} e^{\text{weight}_k}$$

Finally, the weight for each sample is multiplied by each feature of this sample transforming the data as follows:

$$\text{Trn}_i = \text{Trn}_i . \text{Softmax}(\text{weight}_i)$$

3.2 Cross-Attention Applied Between Training and Testing Dataset

In the Transformer model, the idea of cross-attention involves looking at the entire source sequence (training dataset) when generating each token in the target sequence (testing dataset).

In our model, cross attention compares the vectors that represent samples in the testing dataset with those in the training dataset. By calculating the dot product between these vectors, we can determine how relevant each testing sample is to each training sample, capturing the relationships between the datasets. To ensure that attention scores add up to one and reflect valid probabilities, softmax normalization is applied to the computed weights. Lastly, by scaling the features of each testing sample

with their respective softmax weights, the model can highlight the most important features during prediction.

Cross-attention is performed as follows: Let Testing features (Target excluded) be the testing dataset with y dimesion and TrnX be the training dataset. Samples in the TestX(Target excluded) are represented by vectors Test1, $Test_2$, . . . , $Test_y$, where $Test_i$ is the i-th vector.

Weight for each $Test_X$ sample expression for $Test_i$ is given by:

$$\mathrm{wTest_i} = \sum_{j=1}^{y} \mathrm{Test_i Trn_j} \quad for\ i = 1,2, \ldots \ldots x$$

Softmax for each weight $wTest_j$ is then calculated using the following equation:

$$\mathrm{Softmax(wTest_j)} = e^{\mathrm{wTest_j}} / \sum_{k=1}^{y} e^{\mathrm{wTest_k}}$$

Finally, the weight for each sample is multiplied by each feature of this sample transforming the data as follows:

$$\mathrm{Test_i} = \mathrm{Test_i.Softmax(wTest_i)}$$

The Proposed Model:

This section describes an innovative methodology approached in this paper to experiment with the impact of using data attention points on handcrafted features enhancing SD prediction. Figure 1 introduces the proposed framework. As shown in the figure, the methodology initiates with the consolidation of different Software defect datasets that share the same features into a unified dataset, then comprehensive data prepossessing was conducted. Subsequently, the dataset is split into training and testing to facilitate the implementation of cross-validation techniques. Two distinct experiments were applied to this unified dataset. The first one performed our deep learning model on the split dataset without performing any additional preprocessing steps or data transformations.

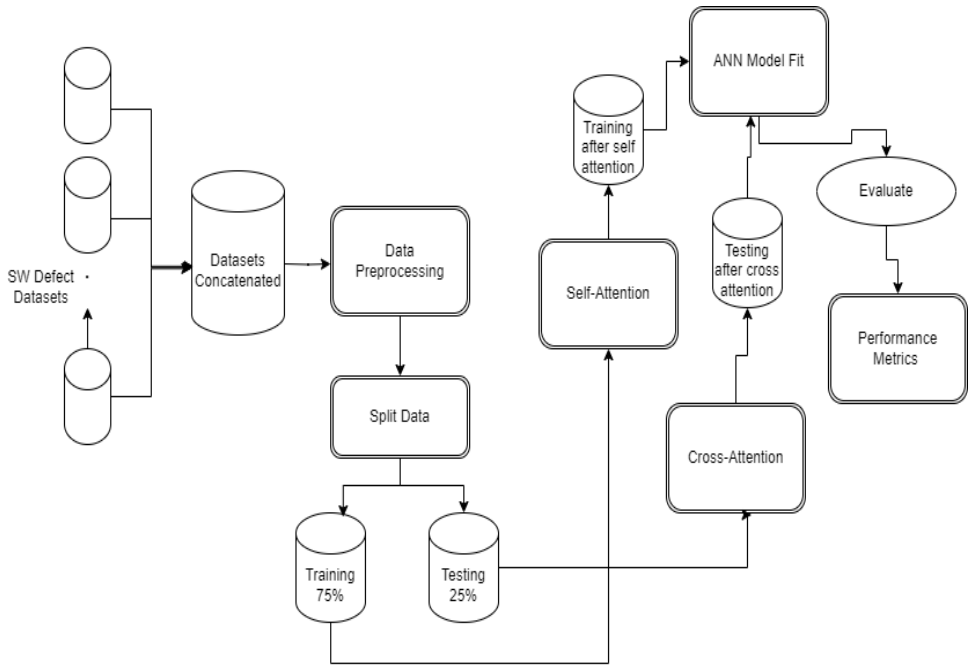

Figure 1 : Proposed Model

In contrast, the second experiment applied our suggested deep learning model after transforming data using data attention points seeking to enhance the performance of the model. Our new approach, named AttDeep combines a deep learning framework (including an input layer, four layers, with a dropout and an output layer) with an attention mechanism. By blending these elements, the model becomes better at recognizing patterns and relationships in the data while preventing overfitting by using dropout regularization.

Additionally, incorporating the attention mechanism boosts the model's capability to concentrate on features leading to enhanced accuracy and interpretability.

The proposed model is explained in the following sections.

Dataset:

There are plenty of well-known open-source Software defect datasets available, which enable testing and evaluating our model. We chose only the dataset that

shared the same metrics for consistency and comparability, so we ended up with twelve datasets from the AEEEM and PROMISE repositories [29]. Each consists of 48 metrics considered as the features. Dataset list with their size shown in table 2.

Table 2. Summary of datasets

Dataset	Samples
JDT~R2~0	2397
JDT~R3~0	3420
JDT~R3~1	3883
JDT~R3~2	2233
jedit~3.	272
jedit~4	367
log4j~1	109
PDE~R2~0	576
PDE~R2~1	761
PDE~R3~0	881
PDE~R3~1	1108
PDE~R3~2	1351

Data in datasets were carefully reviewed to ensure consistency before merging them into one dataset resulting in a total of 16,610 samples. It's worth noting that the dataset has a sparsity ratio of 43.43\%. This ratio is calculated using the formula;

$$SR = 1 - \frac{\text{minority}}{\text{majority}}$$

The dataset was carefully validated to maintain data accuracy. Upon examination, it was confirmed that the dataset consists of data without any textual elements. Additionally, no instances of missing data were detected in the dataset. It's worth mentioning that zeros in this data set hold significance suggesting the absence of bug errors in the target column, and denoting zero values for the features.

Data Pre-Processing:

Data preprocessing is an essential step preceding ML, ensuring quality data for input to facilitate generating reliable results for the output. The data in hand has no null or missing values. Nevertheless, cleaning data involved removing illogical samples such as bugs with no line of codes, and we ended up with 16,605 samples. Given that the dataset does not have the data density feature field which is the required target column, we created a new column called DataDensity that incorporates the required data density information. Data density is calculated as follows:

DataDensity = (Number of Bugs/Line of Code) *1000 (8)

The ready-to-use dataset contains the 48 metrics as features, number of bugs column was dropped as it was substituted by DataDensity which serves as the target column. Features were then normalized using MinMaxm normalizer, to keep consistency between different scales and measurements. It's also essential for improved performance. Finally, splitting the dataset into training and testing is one important step for model evaluation and detecting its ability to generalize. Three-quarters (0.75%) of the data were allocated for training while the remaining quarter (0.25 %) was reserved for testing purposes.

The Experiment:

This section presents the details of the two experiments applied to the dataset. The experiments were accomplished using Python with Anaconda Jupiter on a high-performance workstation with sufficient storage (512 GB SSD) and (32GB) RAM. The host machine operates on Windows 10 with a Core i7 12th Gen Intel(R) CPU (3.5 GHz) with a dedicated GPU.

Deep Learning Model:

The proposed model is a feedforward NN comprising of an input layer, a dropout layer, four hidden layers and an output layer. The input layer is designed with 48

dimensions to accommodate the number of features. After conducting experiments and adjusting the hyperparameters based on the task of predicting software defect density and our selected dataset we settled on a specific structure. This structure comprises four layers with varying numbers of neurons; 20, 100, 50 and 100 respectively. We opted for ReLU activation functions in these layers as they are well-suited for learning models. They are efficient. Ensure that gradients do not vanish during training and backpropagation. These hidden layers follow a pattern known as layer pattern. It involves performing matrix multiplication between the input features and weights adding a bias term and applying the ReLU activation function. As for the output layer, it consists of a neuron with linear activation. This choice is made because our model is designed to handle regression tasks where the output is expected to be a value ranging from 0 to 1. Additionally, we have incorporated a dropout layer, with a rate of 0.3 which serves as a regularization technique. During training this layer, randomly removes a percentage of neurons to prevent overfitting. With the 0.3 rate specified 30% of neurons will be dropped during each training iteration. For the Compilation configuration, we have opted for the Adam optimizer due to its efficiency and effectiveness in handling learning rates and adapting to gradients.

4.4 Evaluation Metrics:

We used loss and Mean Absolute Error (MAE) evaluations to validate the performance of our model for both training and validation datasets. Through applying these metrics we can understand how the models learn and enhance their performance with time. The reason for utilizing these metrics is because they are good at measuring the difference between predicted and real values. A reduction in both loss and MAE values indicates that the model can reduce errors and enhance its capability. Additionally, evaluating how well the model performs on training versus validation datasets aids in identifying overfitting problems and guarantees that the

model can make predictions beyond the training data. We selected these metrics because they are commonly used in ML and due to their simplicity as they are easy to understand making them ideal for evaluating how well predictive models work in areas like predicting software defects.

Hyperparameters:

In our efforts to enhance the learning model for forecasting software defect density we conducted a set of experiments to identify the most suitable hyperparameter configurations.

With a sizable dataset comprising around 16,605 samples, our aim was not just accuracy in predictions but efficiency in computing. After rounds of testing, we established a setup that struck a balance, between predictive precision and computational efficiency. We decided to go with the Adam optimizer because its good at handling sparse gradients and performs well on data with noise. We set the learning rate to 0.1 after trying out values. We found that it consistently led to faster convergence and kept the learning process stable. For building the model, we used mean error as the loss function. Mean absolute error (MAE) for assessing performance, which aligns with the regression aspect of our task. These measures effectively gauge the accuracy of our predictions, which is crucial for predicting defect density across continuous variables. The training process lasted for 30 epochs with a batch size of 300 a duration and quantity that have been empirically shown to enable the model to learn from the data without falling into overfitting. This batch size struck a balance, between efficiency and the models ability to generalize— batches allowed for faster computations but potentially weaker generalization abilities while smaller batches facilitated more precise model updates but required additional time. During the training process, we included validation data to continuously evaluate how well the model performs on data. This helped us choose hyperparameters that led to a model that not only worked well with the training data but also could adapt

effectively to datasets. It was important for us to keep things the same throughout our experiments by using hyperparameter settings. This helped us make comparisons between the models' performances in situations without any interference from different hyperparameters. By sticking to this method, we feel confident in the insights we gained from comparing these experiments, which further supports the strength and trustworthiness of our findings when looking into how attention mechanisms work in learning models for predicting software defect density.

Table 3. Models Results.

Models	**Evaluation Metrics**			
	Training Loss	**Training MAE**	**Testing Loss**	**Testing MAE**
AttDeep	.0612	.0306	.0230	.0283
WithoutAttrntion	.0668	.0718	.0265	.0655

Results and Discussions:

In this section, we will be discussing the outcomes of our experiments. We noticed that the model incorporating the attention mechanism generally performs better than the model without it in terms of both training and validation losses. Results are found in Table 2. The model consistently achieves losses. Exhibits more stable metrics during each epoch. Additionally, we found that the inclusion of an attention mechanism contributes to the generalization of the model as indicated by its validation metrics. On the other hand, the model without attention demonstrates variations in both training and validation losses suggesting robust training.

While it is true that the model with attention requires calculations due to its inclusion of this mechanism, it ultimately leads to convergence and overall performance.

Model Without Attention:

In the first model, we applied the proposed model to the dataset in hand without any data transformation preceding the learning. The loss and MAE values for both the

training and validation sets show a decrease suggesting that the model is learning and improving its performance. The model results show a change in Loss values from 459268.9688 decreases to 0.0668 and MAE decrease from 71.4413 to 0.0718. However, after around epoch 10, the loss and MAE values for the validation set stabilize indicating that there may not be an improvement in the model's performance beyond this point. On the other hand, the training loss and MAE continue to decrease across epochs indicating that the model is still learning from the training data. In terms of generalization, it is noteworthy that the validation loss (0.0265) and MAE (0.0655) values consistently remain lower than their training metrics. This indicates that the model performs well with unseen data without going overfitting to the training data. The fact that both validation and training metrics are close further strengthens our confidence in the model's ability to generalize effectively. The hyperparameters used during training – such as learning rate, batch size and number of epochs – appear reasonable since they lead to a decrease in both training and validation loss/MAE throughout training.

Model with Attention:

The second model (AttDeep) which includes the attention mechanism is showing promising outcomes as it appears to capture patterns in the data. This can be observed through the decreasing loss and MAE. The loss and MAE values for both the training and validation sets are relatively low, indicating the good performance of the model. Additionally, the difference between the training and validation metrics is not very significant, suggesting that the model is not overfitting the training data. Both the loss and MAE values seem to stabilize as the number of epochs increases. Initially, there is a significant decrease in both loss and MAE, but it becomes more gradual over time.

The stability of these metrics suggests that the model has convergence to a certain extent. Further training beyond 30 epochs may not significantly improve performance

and might risk overfitting. The validation metrics including validation loss and MAE are comparable to the training metrics, indicating good generalization of the model to unseen data. The slight differences in validation metrics compared to training metrics suggest a minor degree of overfitting, but overall, the model seems to generalize well to new data. The chosen hyperparameters including the learning rate and dropout rate seem to be reasonable for the task and dataset. The Adam optimizer with a learning rate of 0.1 is commonly used and appears to work effectively in this context. The dropout layer with a rate of 0.3 serves as a regularization technique to prevent overfitting, which is a good practice.

The attention mechanism incorporated into the model aims to capture important relationships in the data that the model can potentially capture and utilize dependencies in the dataset more effectively compared to a model without attention. It demonstrates convergence, reasonable generalization to unseen data, appropriate hyperparameter choices, and the utilization of an attention mechanism to potentially capture important relationships in the dataset. Further analysis and experimentation could help fine-tune the model and potentially enhance its performance for the specific task of predicting software defect density.

Discussion:

After comparing the outcomes of both models, we observed that the deep learning model without attention has slightly higher loss and MAE values compared to the our AttDeep model. It seems like the model might not give the optimal performance on data if it doesn't use the attention technique, unlike our AttDeep model. Also, looking at the validation loss and MAE values it's clear that the model without attention shows higher values compared to the AttDeep model. This suggests that without attention, the model could have a time making predictions on new data possibly showing signs of overfitting on the training data. Both models seem to stabilize in

terms of loss and MAE values as training progresses although it's important to note that the model without attention ultimately yields a higher final loss and MAE. It's worth mentioning that due to its lack of an attention mechanism, the non-attention model might be less complex. However, it's important to consider that the effectiveness of attention mechanisms can vary depending on factors such as data characteristics and task requirements. Overall, while the non-attention model can have reached convergence during training. It appears to exhibit slightly lesser performance compared to the AttDeep model; This is evident from higher loss. Higher MAE values were observed across both training and validation datasets. Hence, incorporating attention mechanisms could prove beneficial, for enhancing model performance in tasks similar to the one in hand.

Contribution:

The research presented in this study makes several important contributions to the field of software defect density prediction and its application in learning. First of all, this study introduces an innovative approach to predicting software defect density by incorporating a data attention mechanism into a deep learning architecture. By exploring NN techniques, this methodology enhances our understanding of how predictive accuracy can be improved in quality assurance tasks. While previous studies have mainly dealt with sequential problems, along with feature extraction using techniques such as CNN and RNN, our work stands out by introducing attention mechanisms alongside handcrafted features in this area. As far as we know, our research is the first to merge attention mechanisms with hand-crafted features for predicting software defect density thus, pushing the boundaries of knowledge in this field. The paper also provides a good empirical evaluation.

Through experimentation and analysis, this research provides evidence that supports the effectiveness of the proposed deep learning model with a data attention point for predicting software defect density. The comparison between our proposed

model (AttDeep) -the one with the attention mechanism and the one without attention mechanisms- offers insights into the performance improvements achieved through the incorporation of attention mechanisms. Through capturing and redistributing knowledge within and between datasets AttDeep shows clarity and predictive precision marking a notable progress in the realm of machine learning and predictive analysis. Moreover, AttDeep model provides improved Model Generalization as the findings of this study demonstrate that the deep learning model with data attention mechanism exhibits generalization capabilities compared to its counterpart without attention. The suggested model shows the potential to aid quality assurance teams in defining defects and reducing their negative impact. Finally, the findings of this study open up avenues, for research in the field of merging the power of deep learning with attention mechanisms in several domains including SDP.

Conclusion:

In conclusion, our research paper provides a novel approach for incorporating data attention mechanisms with deep learning. During the data preprocessing phase, illogical data points were deleted. A new feature column called DataDensity was created to capture information for defect density prediction. Features were normalized to maintain consistency across scales leading to improved model performance. Two experiments were conducted using a deep learning architecture, one with a data attention mechanism and one without. The results consistently showed that the model incorporating the attention mechanism outperformed the model in terms of loss and MAE values across both training and validation datasets. Furthermore, the analysis indicated that integrating the attention mechanism helped improve the model's ability to generalize data by reflecting the content of the data to each part of the dataset using the attention technique. Comparing both models, it was

observed that while both achieved convergence during training the model, AttDeep model exhibited slightly superior performance, with lower loss and MAE values.

This research highlights the benefits of incorporating attention mechanisms into deep learning models across various fields. The findings suggest that integrating attention mechanisms can enhance model performance by capturing and utilizing patterns in the data over time.

Additionally, applying attention mechanisms can be very impactful for areas where dataset is sensitive. Sensitive data are hard to oversample or under sampled. Thus, it's essential to find a way that captures the pattern and reflects it on the whole dataset. To summarize, this study emphasizes the significance of utilizing techniques like data attention points to enhance the abilities of deep learning models, and use it in different fields. Our paper applied the model on SDP, data in other fields must experimented as future work.

References:

[1] M. B. R. Pandit and N. Varma, "A deep introduction to AI based software defect prediction (SDP) and its current challenges," presented at TENCON 2019 – 2019 IEEE Region 10 Conference (TENCON), pp. 284–290, 2019.

[2] P. Mahbub, O. Shuvo, and M. M. Rahman, "Explaining software bugs leveraging code structures in neural machine translation," presented at IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 640–652, 2023.

[3] M. E. Fagan, "Advances in software inspections," *IEEE Transactions on Software Engineering*, vol. SE-12, pp. 744–751, 1986.

[4] S. Kumar, R. K. Singh, and A. K. Maurya, "Software defect prediction: State of the art survey," *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* vol. 11, pp. 32–35, 2022.

[5] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th international conference on software engineering*, pp. 414–423, 2014.

[6] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 46, pp. 1267–1293, 2020.

[7] J. Chen, K. Hu, Y. Yang, Y. Liu, and Q. Xuan, "Collective transfer learning for defect prediction," *Neurocomputing*, vol. 416, pp. 103–116, 2020.

[8] M. Wen, R. Wu, and S. Cheung, "How well do change sequences predict defects? sequence learning from software changes," *IEEE Transactions on Software Engineering*, vol. 46, pp. 1155–1175, 2020.

[9] H. Chen, X. Jing, Z. Li, D. Wu, Y. Peng, and Z. Huang, "An empirical study on heterogeneous defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. 47, pp. 2803–2822, 2021.

[10] F. Matloob, T. M. Ghazal, N. Taleb, S. Aftab, M. Ahmad, M. A. Khan, S. Abbas, and T. R. Soomro, "Software defect prediction using ensemble learning: A systematic literature review," *IEEE Access*, vol. 9, pp. 98754– 98771, 2021.

[11] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Machine learning- based software defect prediction for mobile applications: A systematic literature review," *Sensors (Basel, Switzerland)*, vol. 22, 2022.

[12] E. A. Felix and S. Lee, "Integrated approach to software defect prediction," *IEEE Access*, vol. 5, pp. 21524–21547, 2017.

[13] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning". *MIT press*, 2016.

[14] W. Zhu, L. Xie, J. Han, and X. Guo, "The application of deep learning in cancer prognosis prediction," *Cancers*, vol. 12, 2020.

[15] F. Murat, Özal Yıldırım, M. Talo, U. Baloglu, Y. Demir, and U. Acharya, "Application of deep learning techniques for heartbeats detection using ecg signals-analysis and review," *Computers in biology and medicine*, vol. 120, p. 103726, 2020.

[16] M. Salvi, U. Acharya, F. Molinari, and K. Meiburger, "The impact of pre- and post-image processing techniques on deep learning frameworks: A comprehensive review for digital pathology image analysis," *Computers in biology and medicine*, vol. 128, p. 104129, 2020.

[17] A. Thompson, A. Jammal, and F. Medeiros, "A review of deep learning for screening, diagnosis, and detection of glaucoma progression," *Translational Vision Science Technology*, vol. 9, 2020.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[19] W. Zhu, J. Li, Z.-Y. An, and Z. Hua, "Mutiscale hybrid attention transformer for remote sensing image pansharpening," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 61, pp. 1–16, 2023.

[20] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[21] S. Ding, S. Qu, Y. Xi, and S. Wan, "Stimulus-driven and concept-driven analysis for image caption generation," *Neurocomputing*, vol. 398, p. 520–530, 2020.

[22] S. Ji, D. Yu, C. Shen, W. le Li, and Q. Xu, "Landslide detection from an open satellite imagery and digital elevation model dataset using attention boosted convolutional neural networks," *Landslides*, vol. 17, pp. 1337– 1352, 2020.

[23] W. Cai and Z. Wei, "Remote sensing image classification based on a cross-attention mechanism and graph convolution," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2020.

[24] W. Tao, C. Li, R. Song, J. Cheng, Y. Liu, F. Wan, and X. Chen, "Eeg-based emotion recognition via channel-wise attention and self-attention," *IEEE Transactions on Affective Computing*, vol. 14, pp. 382–393, 2023.

[25] Boehm, B. W. (2002). "Software engineering economics" (pp. 641-686). Springer Berlin Heidelberg.

[26] N. Fenton and S. Pfleeger, "Software metrics, a rigorous and practical approach" *Thomson International* (Comp. Press) ,1996

[27] N. Dayyala, K. Walstrom, K. Bagchi, and G. Udo, "Factors impacting defect density in software development projects," *The International Journal of Information Technologies and Systems Approach*, vol. 15, pp. 1–23, 2022.

[28] J. Kelly, J. Sherif, and J. Hops, "An analysis of defect densities found during software inspections," J. *System Software.*, vol. 17, pp. 111–117, 1992. vol 4, 2016 9

[29] F. Alghanim, M. Azzeh, A. El-Hassan, and H. Qattous, "Software defect density prediction using deep learning," *IEEE Access*, vol. 10, pp. 114629– 114641, 2022.

[30] H. B. Yadav and D. Yadav, "A fuzzy logic based approach for phase-wise software defects prediction using software metrics," *Information and Software Technology.*, vol. 63, pp. 44–57, 2015.

[31] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv*:1409.0473, 2014.

[32] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," presented in International conference on machine learning, pp. 2048–2057, PMLR, 2015.

[33] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv*:1409.1259, 2014.

[34] C. Yan, Y. Tu, X. Wang, Y. Zhang, X. Hao, Y. Zhang, and Q. Dai, "Stat: Spatial-temporal attention mechanism for video captioning," *IEEE Transactions on Multimedia*, vol. 22, pp. 229–241, 2020.

[35] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 1441–1450, 2019.

[36] W. Wen, R. Zhang, C. Wang, C. Shen, M. Yu, S. Zhang, and X. Gao, "A cross-project defect prediction model based on deep learning with self-attention," *IEEE Access*, vol. 10, pp. 110385–110401, 2022.

[37] W. Zheng, L. Tan, and C. Liu, "Software defect prediction method based on transformer model," in 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), pp. 670–674, 2021.

[38] S. Qiu, S. Wang, X. Tian, M. Huang, and Q. Huang, "Visualization-based software defect prediction via convolutional neural network with global self-attention," in 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), pp. 189–198, 2022.

[39] T. B. Shahi, C. Sitaula, A. Neupane, and W. Guo, "Fruit classification using attention-based mobilenetv2 for industrial applications," *PLoS ONE*, vol. 17, 2022.

[40] B. Zhang, D. Xiong, and J. Su, "Neural machine translation with deep attention," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 154–163, 2020.

[41] T. Zhang, Q. Du, J. Xu, J. Li, and X. Li, "Software defect prediction and localization with attention-based models and ensemble learning," in 2020 27th Asia-Pacific Software Engineering Conference (APSEC), pp. 81–90, 2020.

[42] O, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning-based software defect prediction. ***Neurocomputing***, 385, 100-110.

[43] Mri, S., & Sinz, C. (2020). Deep learning for software defect prediction: A survey. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (pp. 209-214).

[44] Kaur, A., Malhotra, R., & Singh, Y. (2021). "Deep Learning Techniques for Software Defect Prediction: A Survey." Journal of Systems and Software, 175, 110905. DOI: 10.1016/j.jss.2021.110905