On Calibration of Pre-trained Code Models

Zhenhao Zhou
School of Computer Science
Fudan University
Shanghai, China
zhenhaozhou21@m.fudan.edu.cn

Chaofeng Sha*
School of Computer Science
Fudan University
Shanghai, China
cfsha@fudan.edu.cn

Xin Peng
School of Computer Science
Fudan University
Shanghai, China
pengxin@fudan.edu.cn

ABSTRACT

Pre-trained code models have achieved notable success in the field of Software Engineering (SE). However, existing studies have predominantly focused on improving model performance, with limited attention given to other critical aspects such as model calibration. Model calibration, which refers to the accurate estimation of predictive uncertainty, is a vital consideration in practical applications. Therefore, in order to advance the understanding of model calibration in SE, we conduct a comprehensive investigation into the calibration of pre-trained code models in this paper. Our investigation focuses on five pre-trained code models and four code understanding tasks, including analyses of calibration in both in-distribution and out-of-distribution settings. Several key insights are uncovered: (1) pre-trained code models may suffer from the issue of over-confidence; (2) temperature scaling and label smoothing are effective in calibrating code models in in-distribution data; (3) the issue of over-confidence in pre-trained code models worsens in different out-of-distribution settings, and the effectiveness of temperature scaling and label smoothing diminishes. All materials used in our experiments are available at https://github.com/queserasera22/Calibration-of-Pretrained-Code-Models.

CCS CONCEPTS

• **Software and its engineering → Software development techniques**; **Software verification and validation**.

KEYWORDS

Pre-trained Code Models, Model Calibration, Model Reliability

ACM Reference Format:
Zhenhao Zhou, Chaofeng Sha, and Xin Peng. 2024. On Calibration of Pre-trained Code Models. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639126

1 INTRODUCTION

Over the past few years, the application of pre-trained models has led to remarkable advancements in the Software Engineering (SE) field. Pre-trained code models, such as CodeBERT [9] and CodeT5 [44], have been proposed and widely used for source code learning. Built on Transformer [43] architecture, these models have excelled at various code intelligence tasks [49] [33].

Despite the remarkable success of pre-trained models, they have faced criticism due to their opacity [7]. It is extremely difficult to understand how these large models arrive at their predictions, which has raised concerns around the reliability of their outputs. When integrating pre-trained models into broader systems, it is crucial for these models not only to be accurate but also to express their uncertainty precisely, so that their errors can be appropriately anticipated and accommodated. However, in the SE field, existing research has largely focused on improving model performance, while giving relatively little attention to other essential aspects like model calibration, which is critical for comprehending the reliability and uncertainty of models.

Model calibration refers to the degree of agreement between the predicted probabilities of a model and the empirical likelihoods [10] [28], which is a critical aspect that reflects whether the model can be trusted. In other words, the output probability of well-calibrated model provides an accurate estimate of the probability that a given event will occur. For instance, when an event has a probability of occurrence of 80%, a model is deemed as well-calibrated if it predicts the event with a probability of 80%. In contrast, if the predicted probability exceeds or falls below 80%, the model is considered over-confident or under-confident, respectively. In this context, the prediction confidence of a model refers to its predicted probability on the event. Although large models may lack interpretability due to their opacity and complexity, calibration can help these models at least "know what they don't know" [7] [35], leading to more accurate and reliable predictions.

Model calibration has already gained considerable attention in the area of computer vision (CV) [10] [28] and natural language processing (NLP) [7] [35] [5] [25] in recent years. And a number of techniques have been proposed to address the problem of model miscalibration, such as temperature scaling [10] and label smoothing [40]. However, the understanding of model calibration in the field of SE remains limited. Although Li et al. [24] have investigated model calibration for deep neural network in operational domains, their focus is on traditional models applied to CV and NLP tasks. There is still insufficient knowledge regarding the calibration of pre-trained code models, despite the increasing use of these models in the field of SE.

Ensuring the calibration of pre-trained code models is a crucial consideration when deploying such models in practical applications, especially in high-stakes settings. For instance, in SE filed, pre-trained code models can serve as software vulnerability detectors. In this context, a well-calibrated model is necessary as it allows

*Corresponding author.

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04
https://doi.org/10.1145/3597503.3639126

accurate estimation of prediction uncertainty. When the model is uncertain in its prediction (displaying low confidence), it provides an opportunity for a supervisor to intervene by conducting a human review to avoid potential errors. Conversely, if the model is miscalibrated and exhibits wrong confidence, the supervisor remains unaware of predictions that are more likely to be erroneous, which may lead to undesirable consequences. Therefore, the calibration of pre-trained code models is essential to improve the reliability of such models and to mitigate potential risks associated with their usage in real-world scenarios.

In this study, we present a systematic analysis of the calibration of pre-trained code models, considering both in-distribution (ID) and out-of-distribution (OOD) settings. Specially, we evaluate the calibration of five different pre-trained code models, namely CodeBERTa [46], CodeBERT [9], GraphCodeBERT [12], CodeT5 [44] and UniXcoder [11]. These code models are based on different architectures and accept different input modalities. Our calibration experiments are conducted on four code understanding tasks and a total of six datasets. Furthermore, we investigate the effectiveness of the two most widely used and representative calibration methods on pre-trained code models, namely temperature scaling [10] and label smoothing [40]. Additionally, we simulate three different distribution shifts in code data to investigate the calibration of pre-trained code models and effectiveness of calibration methods in OOD settings.

Several key insights are drawn from our experimental results. Firstly, pre-trained code models are not always well-calibrated and suffer from the issue of over-confidence in ID setting. And pre-trained models achieving higher accuracy do not always show better calibration. Secondly, temperature scaling and label smoothing are effective to calibrate code models in ID setting, which could reduce calibration error significantly. Finally, we find the issue of over-confidence of pre-trained code models gets worse in OOD settings. Temperature scaling and label smoothing are less effective when there is a distribution shift. These findings devote a better understanding on the calibration of pre-trained code models and call for more attention to improve the calibration of pre-trained code models.

In general, the contributions of our work are as follows:

- We investigate the calibration of pre-trained code models. To the best of our knowledge, this is the first empirical study on the calibration of pre-trained code models in the SE field.
- We make a comprehensive analysis of the calibration of five pre-trained code models in both ID and OOD scenarios.
- We perform a thorough evaluation of the effectiveness of two calibration methods, temperature scaling and label smoothing, for pre-trained code models in both ID and OOD scenarios.

2 BACKGROUND

In this section, we briefly present some preliminaries of our work, including model calibration, calibration measures, and calibration methods.

2.1 Model Calibration

Model calibration refers to the degree to which the predicted probabilities of a model align with the true likelihood [10] [13] [42]. In a classification problem with k classes, let x be the input instance with label y, and let $(a_1, ..., a_k)$ represent the output probabilities of a model on the k classes, where $\sum a_i = 1$. The model prediction, denoted as $\hat{y}$, is identified as $\arg\max_i(a_i)$, while its corresponding predicted probability or confidence, denoted as $\hat{p}$, is represented as $a_{\hat{y}}$. It is desirable for the model confidence to reflect true likelihood of the prediction. Specifically, a model is considered well-calibrated if for every class i:

$$\mathbb{P}(y = i | a_i = p) = p, \quad \forall p \in [0, 1], \quad \forall i \in [1, k] \tag{1}$$

where $\mathbb{P}$ denotes the true probability of the event. However, it is challenging to evaluate $P(y = i|a_i = p)$. Therefore, an empirically defined well-calibrated model satisfies:

$$\mathbb{P}(y = \hat{y} | \hat{p} = p) = p, \quad \forall p \in [0, 1] \tag{2}$$

2.2 Calibration Measures

There have been some metrics proposed to measure the uncertainty and calibration of models, such as Expected Calibration Error (ECE) [30], Maximum Calibration Error (MCE) [30],Brier Score [3], Negative log likelihood(NLL) [10], and other theoretic-motivated metrics [41] [13]. Moreover, visual representation of model calibration can be achieved through the use of reliability diagrams [6]. We employ ECE and reliability diagrams for the main analyses in our work for their popularity and simplicity.

ECE is a widely adopted metric to evaluate the calibration of probabilistic predictions. ECE is defined as the expected value of the absolute difference between the accuracy and confidence of a model, where confidence is defined as the predicted probability of the predicted class. Specifically, the confidence interval from 0 to 1 is partitioned equally into M subintervals, referred to as bins, with each bin having a size of $1/M$. And the model predictions are then discretized into the M bins based on the corresponding prediction confidence. Let B_m be the set of instances whose predicted probabilities fall into the m-th bin $(\frac{m-1}{M}, \frac{m}{M}]$.The ECE is calculated as follows:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} \left|\text{acc}(B_m) - \text{conf}(B_m)\right| \tag{3}$$

where n is the total number of instances, $\text{acc}(B_m)$ is the average empirical accuracy of the model on the instances in B_m, and $\text{conf}(B_m)$ is the average predicted probability of the model for the instances in B_m. Fig. 1 illustrates an example of calculating ECE, where model predictions are divided into two bins.

Reliability diagrams are visualization of model calibration, which are represented as histograms with pairs of $(\text{conf}(B_m), \text{acc}(B_m))$. It should be noted that reliability diagrams do not show the proportion of samples in a given bin, hence additional confidence distribution diagrams are typically required. Fig. 2 illustrates an example of confidence distribution diagrams and reliability diagrams, where the confidence values are divided into ten bins in the interval [0,1]. The fraction (Frac) of samples in each bin among total samples is shown in the confidence distribution diagrams. The reliability diagram presents the average accuracy in a given bin as a blue

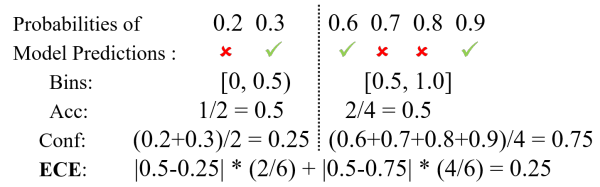


Figure 1: An illustration of the calculation of ECE. Note that the figure uses 2 bins for simplicity, while we use 10 bins in our experiments.

bar, while the calibration gap between the average confidence and average accuracy is depicted in red. If the red bar exist upon the blue bar, it means the confidence of the model is higher than the average accuracy, indicating over-confidence. Conversely, if the red bar is beneath the blue bar (presented as purple), the model exhibits under-confidence.

2.3 Calibration Methods

In order to address the problem of miscalibration in neural networks, various techniques have been proposed. These methods can be broadly categorized into the following classes: post-hoc calibration [10], regularization [36, 40], data augmentation [15] [22] [35], averaging multiple predictions [23] [45] and learnable methods [48] [51]. In this paper, we focus on two simple but effective calibration methods, temperature scaling and label smoothing, which are also the most representative and widely-used methods.

2.3.1 Temperature Scaling. Temperature scaling [10] is a post-hoc calibration technique that re-scales the output probabilities of a classifier model using a single temperature parameter T ($T > 0$). The softmax function with temperature scaling can be defined as:

$$p_T(x) = softmax_T(z)_i = \frac{\exp(z_i/T)}{\sum_{j=1}^{k} \exp(z_j/T)}, \quad i = 1, ..., k \tag{4}$$

where z is the unnormalized logits output by the model. This function leads to more uniform probabilities over the classes with $T > 1$, while assigns more probability mass to one class if $T < 1$.

The value of the temperature parameter T is determined through training. Temperature scaling can be formulated as an optimization problem to find the optimal temperature parameter that minimizes calibration error, as expressed in Eq.5:

$$T_{opt} = \arg\min_{T>0} \quad \mathbb{E}_{x,y \in D_{valid}}[L_c(p_T(x), y)] \tag{5}$$

where x and y denote the input and labels on the validation set D_{valid}, and L_c represents the calibration error targeted for optimization.

2.3.2 Label Smoothing. Label smoothing [7] is a regularization technique that introduces a small amount of uncertainty into the training targets by smoothing the one-hot encoding labels to distribute some of the probability mass from the correct class to the other classes. The smoothed labels are defined as follows:

Table 1: Statistics of Datasets

Tasks	Datasets	#Train	#Validate	#Test	Language
Functionality Classification	POJ104	33,280	8,320	10,400	C / C++
	Java250	48,000	12,000	15,000	Java
	Python800	153,600	38,400	48,000	Python
Code Clone Detection	BigCloneBench	90,102	4,000	4,000	Java
Defect Detection	Devign	21,854	2,732	2,732	C
Exception Type	-	18,480	2,088	10,348	Python

$$\tilde{y}_i = \begin{cases} 1 - \varepsilon, & i = y \\ \varepsilon/(k-1), & i \neq y \end{cases} \tag{6}$$

where $\tilde{y}_i$ is the smoothed label for class i, y is the true label, and α is the smoothing parameter. The smoothing parameter α controls the amount of smoothing applied to the labels. When $0 < \alpha < 1$, some probability mass is shifted from the correct class to the other classes, which encourages the model to be less confident in its predictions.

3 EXPERIMENTAL SETUP

In this section, we describe briefly the settings in our experiments, including models, tasks and datasets, evaluation metrics, and other settings.

3.1 Models

In our experiments, we evaluate the calibration of five pre-trained code models, namely CodeBERTa [46], CodeBERT [9], GraphCodeBERT [12], CodeT5 [44] and UniXcoder [1] [11]. For the sake of contrast, we also assess the calibration of two non-pre-trained models, TextCNN [20] and ASTNN [50], which are representative traditional deep learning approaches utilized for classification tasks in the field of NLP and SE.

3.2 Tasks and Datasets

It is worth noting that previous investigations into model calibration in the fields of NLP and CV have typically focused on classification tasks [5, 7, 10, 25, 28, 35], which inherently allow for the assessment of accuracy and probability. In alignment with these studies, our experiments revolve around four classification tasks within the SE domain, namely functionality classification, code clone detection, defect detection, and exception type prediction. There are six datasets totally in our experiments, which involves different programming languages. The statistics of the datasets used in this work are presented at Table 1.

3.2.1 Functionality Classification. The objective of this task is to classify given code examples into appropriate classes based on their functionality. In this study, we conduct experiments on three well-known datasets, namely POJ104 [29], Java250 and Python800 [37], which contain code examples collected from different online judge websites. Specifically, there are 104, 250, and 800 problem sets, which are composed of submissions written in C/C++, Java, and Python languages, respectively.

[1]For CodeT5 and UniXcoder, we utilized the base versions of the models in our experiments.

Table 2: Experimental Results of Accuracy(%) and ECE (%)

Task		Functionality Classification			Clone Detection	Defect Detection	Exception Type	Average
Model	Metric	POJ104	Java250	Python800				
TextCNN	Acc	98.33	96.37	97.71	89.75	62.63	64.31	84.85
	ECE	**0.53**	**0.38**	**0.34**	2.80	7.46	**7.30**	3.14
ASTNN	Acc	96.04	96.52	94.77	97.53	59.99	52.89	82.96
	ECE	2.10	1.76	0.76	**1.27**	9.76	9.39	4.17
CodeBERTa	Acc	98.41	97.59	98.38	96.82	62.52	72.44	87.69
	ECE	1.12	0.90	0.71	2.71	9.83	20.62	5.98
CodeBERT	Acc	98.49	98.15	98.64	96.75	63.25	79.85	89.19
	ECE	1.17	0.87	0.71	3.11	12.14	17.21	5.87
GraphCodeBERT	Acc	**98.85**	**98.29**	**98.72**	96.53	64.09	80.66	89.52
	ECE	0.74	0.41	0.48	3.42	8.82	12.31	4.36
CodeT5	Acc	98.48	98.09	98.44	**97.88**	63.91	81.27	89.68
	ECE	1.05	0.61	0.78	2.03	9.72	15.44	4.94
UniXcoder	Acc	98.40	98.21	98.72	97.03	**64.79**	**81.72**	89.81
	ECE	1.00	0.75	0.71	2.89	**6.43**	15.13	4.49

3.2.2 Code Clone Detection. The objective of code clone detection is to predict whether a pair of code snippets are similar to each other. Following the work of Niu et al.[33], we use a well-known benchmark, BigCloneBench [38], which is a collection of various Java methods that includes both cloned and non-cloned pairs. In view of the large scale of the original dataset, we use directly the dataset provided by Yang et al. [47], which samples around 10% data from CodeXGLUE benchmark [27].

3.2.3 Defect Detection. The defect detection task aims to identify whether the given code snippets is vulnerable. Following the work of Niu et al.[33], we use the dataset named Devign [52], which is also included as part of the CodeXGLUE benchmark [27]. Devign contains functions extracted from two open-source C libraries, FFmpeg and Qemu, and the data are labeled manually.

3.2.4 Exception Type. Likewise, following the work [33], we conduct our experiments on the task of exception type, using the dataset from Kanade et al. [18]. This dataset comprises a collection of 20 distinct exception types that are commonly encountered in Python programming, sourced from the GitHub dataset. Each example in the dataset is a function that includes an except clause for a specific exception type, with the exception replaced by a special "hole" token. This task is a multi-class classification problem, where the objective is to predict the original exception type.

3.3 Evaluation Metrics

In our experiments, we employ the accuracy (Acc) to measure the overall performance of pre-trained code models. For the evaluation of calibration, we use the most widely used calibration metric, Expected Calibration Error (ECE) [30], following previous work [7] [10] [28]. Additionally, we use reliability diagrams [6] [32] to visualize the difference between the average predicted accuracy and the confidence of the models.

Following the settings in previous study[7], we set the number of bins $M = 10$ for the experiments in this paper.

3.4 Other Settings

Our code implementations are based on the official public repositories of the involved code models. The hyperparameters are set under the guide of previous studies [33] [47] [27]. To optimize the models, we employ the Adam algorithm [21] and set an early stopping condition to terminate training if no improvement is observed after 5 epochs. We run all experiments on a Linux machine with a CPU of Intel(R) Core(TM) i9-10980XE @ 3.00GHz and four 24GB GPUs of NVIDIA GeForce RTX 3090. Further details regarding our implementation can be accessed in our online materials.

4 EXPERIMENTAL RESULTS AND ANALYSIS

This section introduces the results of our experiments and presents the analysis of the results. We aim to answer the following research questions (RQ) through experiments:

RQ1: Are pre-trained code models well-calibrated?

RQ2: How effective are existing calibration methods on pre-trained code models?

RQ3: How about the calibration of pre-trained code models and the effectiveness of calibration methods in OOD settings?

4.1 Model Calibration (RQ1)

We conduct experiments to evaluate the calibration of various code models on four downstream SE tasks.

As depicted in Table 2, the pre-trained code models demonstrate better overall performance across the evaluated tasks. Specifically, in both functionality classification and code clone detection tasks, all the models achieve a high accuracy while maintaining a low calibration error. This observation may be attributed to the simplicity of the two tasks, where all models perform reasonably well with minimal discernible differences. However, in the defect detection

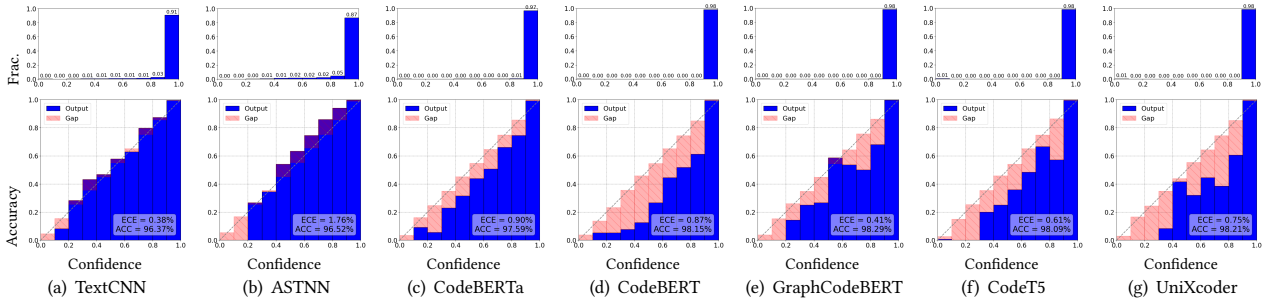


Figure 2: Confidence distribution (top row) and reliability diagrams (bottom row) for different code models on Functionality Classification (Java250).

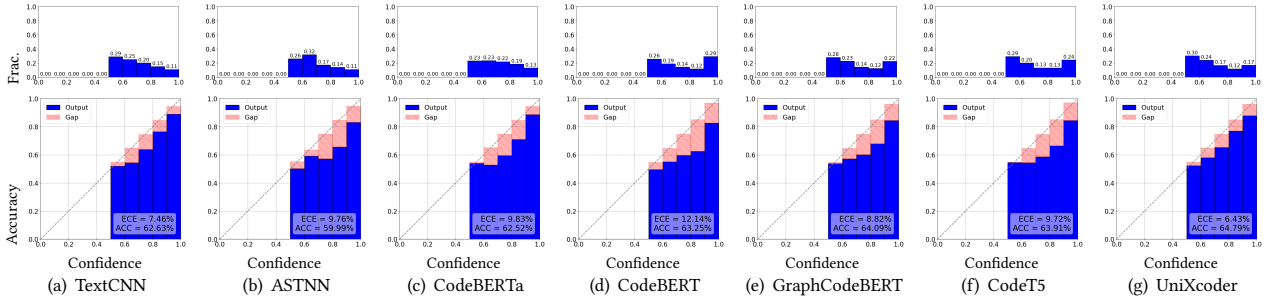


Figure 3: Confidence distribution (top row) and reliability diagrams (bottom row) for different code models on Defect Detection.

task, especially for the non-pre-trained code models, the performance is notably poorer, with an accuracy of approximately 60%. As for the task of exception type prediction, there is a significant variation in performance among different models, with UniXcoder and CodeT5 showing superior overall performance. Nonetheless, the code models exhibit poor calibration in the defect detection and exception type prediction tasks.

In terms of calibration, Table 2 provides several noteworthy observations. Firstly, it is evident that non-pre-trained models, TextCNN and ASTNN, achieve the lowest average ECE across four SE tasks, suggesting that they exhibit superior calibration than pre-trained models. Specifically, ASTNN exhibits the best calibration on the clone detection task, while TextCNN demonstrates the best calibration on the functionality classification and exception type tasks. Secondly, we observe that higher accuracy does not always correlate with better calibration. For instance, UniXcoder achieves better average accuracy than GraphCodeBERT, while it shows inferior calibration. Additionally, achieving the best performance and calibration simultaneously is challenging for most models, except for UniXcoder, which outperforms other models on the defect detection task while maintaining good calibration. These results suggest that it is not enough to focus solely on improving accuracy, and more attention needs to be paid to the calibration of pre-trained code models, in order to ensure reliable uncertainty estimates and more reliable model predictions.

Furthermore, we utilize confidence distribution and reliability diagrams to visually analyze the disparity between the accuracy and confidence of code models. Specifically, Fig.2 presents the diagrams of different models on the Java250 dataset in functionality classification task. Upon observing the confidence distribution plots, it is apparent that the predicted values of all code models predominantly fall within the confidence interval of (0.9,1], suggesting that the task may be relatively uncomplicated and that the models generate highly confident predictions. Consequently, the ECE values of the models are mainly influenced by this high-confidence region of the data. From the reliability diagrams, we find that pre-trained code models exhibit a larger discrepancy between the confidence and accuracy in comparison to TextCNN and ASTNN, the non-pre-trained models. This discrepancy is evident from the presence of larger red gap areas, indicating poorer calibration of pre-trained models relative to non-pre-trained models in functionality classification task and a higher likelihood of over-confident predictions. It is worth mentioning that the calibration of code models is consistent across different datasets on functionality classification, clone detection, and exception type tasks, and further details can be found in our online materials.

The calibration performance of the code model differs on the defect detection task compared to the previously mentioned cases. Fig.3 shows confidence distribution and reliability diagrams on defect detection task. From the confidence distribution plots, it can be observed that the predicted values of different models are not concentrated in the confidence interval of (0.9,1], as seen in other tasks. Instead, they are scattered in the interval of (0.5,1]. This may be attributed to the relatively challenging nature of the task, as

Table 3: Experimental Results of Accuracy (%)and ECE (%) Using Temperature Scaling(TS) and Label Smoothing(LS)

Task		Clone Detection			Defect Detection			Exception Type		
Model	Metric	Original	TS	LS	Original	TS	LS	Original	TS	LS
TextCNN	Acc	89.75	89.75	90.15	62.63	62.63	63.40	64.31	64.31	65.00
	ECE	2.80	2.55	5.46 ↑	7.46	2.80	1.43	7.30	4.41	5.98
ASTNN	Acc	97.53	97.53	97.43	59.99	59.99	61.05	52.89	52.89	53.93
	ECE	1.27	0.44	4.94 ↑	9.76	3.95	6.56	9.39	3.65	0.94
CodeBERTa	Acc	96.82	96.82	96.90	62.52	62.52	61.57	72.44	72.44	73.78
	ECE	2.71	2.02	1.84	9.83	4.31	11.89 ↑	20.62	11.62	10.26
CodeBERT	Acc	96.75	96.75	97.03	63.25	63.25	62.23	79.85	79.85	79.41
	ECE	3.11	2.08	1.30	12.14	5.97	11.27	17.21	11.68	6.44
GraphCodeBERT	Acc	96.53	96.53	96.50	64.09	64.09	62.63	80.66	80.66	81.36
	ECE	3.42	2.86	1.83	8.82	2.86	15.67 ↑	12.31	3.80	5.63
CodeT5	Acc	97.88	97.88	97.12	63.91	63.91	62.15	81.27	81.27	80.69
	ECE	2.03	1.17	2.75 ↑	9.72	4.02	17.42 ↑	15.44	12.30	2.88
UniXcoder	Acc	97.03	97.03	97.22	64.79	64.79	63.91	81.72	81.72	81.42
	ECE	2.89	2.09	2.30	6.43	1.99	5.72	15.13	9.97	5.60

evidenced by the comparatively lower accuracy of all models on this task. In the reliability diagrams, the distribution of discrepancies between model confidence and expected accuracy is similar across different models, with the differences in overall calibration mainly stemming from the distribution of model confidence. Notably, pretraiend code models tend to produce higher confidence, which causes miscalibration.

Summary. Non-pre-trained models exhibit superior calibration, while pre-trained code models suffer from the issue of over-confidence to some extent. Additionally, it is not enough to focus solely on improving accuracy, for pre-trained model achieving higher accuracy do not always show better calibration.

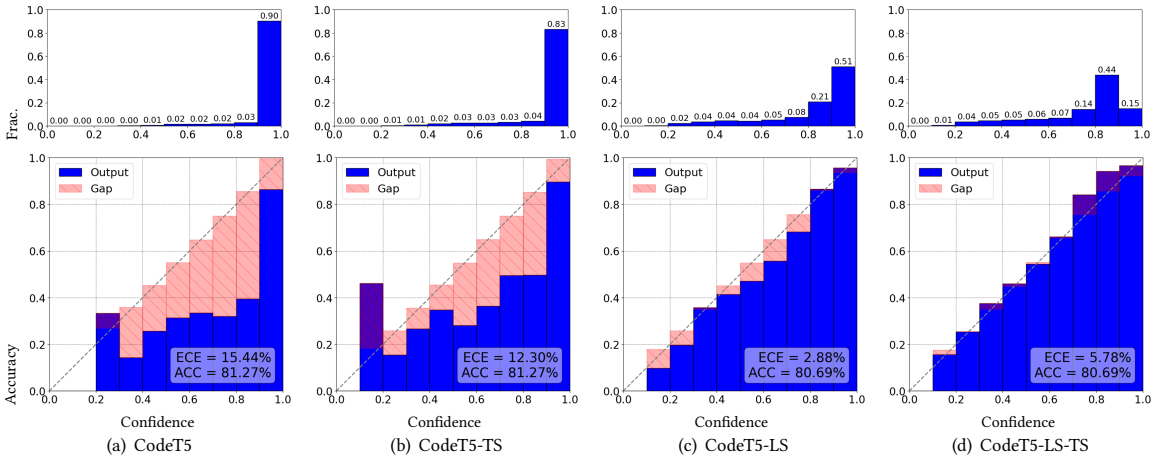
4.2 Calibration Methods (RQ2)

Since pre-trained code models may be over-confident, it is necessary to calibrate these models before deploying them in practical applications. As previously mentioned, our investigation in this study centers on the two most widely used calibration methods, namely temperature scaling and label smoothing. We begin by evaluating the efficacy of these methods in the ID setting. Given the low ECE observed in the functionality classification task above, there may be limited room for improvement in terms of calibration. Therefore, we conduct experiments on other tasks, namely clone detection, defect detection, and exception type.

In our experiments, we follow the settings described in the paper by Desai and Doshi [7]. For temperature scaling, we initialize the value of the temperature parameter T with 1.5, and then optimize the value of T to minimize the NLL loss on the validation sets. We then scale the final output of the model using this optimized value of T. As for label smoothing, we set the value of the label smoothing parameter α to 0.1 in our experiments. The experimental results of the model calibrated with temperature scaling and label smoothing are presented in Table 3.

From the table, it can be observed that temperature scaling is a simple yet effective calibration method for ID data, which could significantly reduce the ECE of the model's predictions. As presented in the table, the accuracy of the model remains unchanged after calibrated with temperature scaling. This can be attributed to the fact that temperature scaling merely re-scales the logits of the model without modifying the final predicted results. Furthermore, we notice that the learned temperature parameter T typically falls within the range of [1,2] in our experiments, which leads to more uniform probabilities over the classes. This indicates that code models exhibit varying degrees of over-confidence phenomenon. And temperature scaling mitigates the issue of miscalibration through reducing the confidence of code models.

Compared to temperature scaling, label smoothing may potentially be more effective for model calibration, albeit with some potential instability. Firstly, we can observe that models calibrated with label smoothing exhibit small fluctuations in accuracy. The uncertainty incorporated in label smoothing may impact the performance of models. Secondly, label smoothing is beneficial for mitigating the issue of over-confidence in code models. However, it may also result in under-confidence in model predictions. As showed in the table, label smoothing is effective to reduce the ECE of code models in exception type task, while it may lead to an increase in ECE for clone detection and defect detection tasks (indicated by ↑ in the table). We also conduct experiments with more



(a) CodeT5 (b) CodeT5-TS (c) CodeT5-LS (d) CodeT5-LS-TS

Figure 4: Confidence distribution (top row) and reliability diagrams (bottom row) for CodeT5 using Temperature Scaling(TS), Label Smoothing(LS) and both(LS + TS) on the task of Exception Type. The CodeT5 models with TS or LS exhibit good calibration, whereas the combination of LS and TS may lead to over-calibration, resulting in potential under-confidence in predictions.

values of α (available in our online materials). We find that the calibration is heavily influenced by this parameter, suggesting that the choice of an appropriate value of α is crucial for achieving effective model calibration.

We take the performance of the CodeT5 model on the exception type task as an example to investigate the effects of the two calibration methods. Fig. 4 presents the confidence distribution and reliability diagrams of the calibrated CodeT5 model after temperature scaling and label smoothing. From the figures, it can be observed that both calibration methods result in a smaller proportion of model outputs with confidence in the interval (0.9,1], and the discrepancy between the model's confidence and average accuracy is further reduced. This indicates that both methods effectively alleviate the issue of over-confidence in the pre-trained code models. However, the combination of these two calibration methods does not result in additive calibration improvement. This is because both methods are intended to address the problem of model over-confidence, and applying them together may lead to over-calibration, causing the model to be under-confident. This phenomenon is evident in Fig. 4(d), where the model's output accuracy is higher than the confidence, resulting in an increase in ECE.

> **Summary**. Both TS and LS are effective in mitigating the issue of over-confidence in model predictions, improving the calibration of the code models. However, over-calibration may cause the model to be under-confident.

4.3 Out-Of-Distribution Calibration (RQ3)

Distribution shift, wherein the test set follows a different distribution from the training set, often occurs in real-world scenarios after a model has been deployed. This phenomenon poses a longstanding challenge, as it may lead to unexpected performance degradation and unreliable predictions[16][2]. Therefore, it is of utmost importance for a model to provide accurate uncertainty estimates not only for ID data but also for OOD data, especially in safety-critical applications.

Table 4: Statistics of Out-Of-Distribution Datasets

Dateset	OOD Type	#Train	#Validate	#OOD Test
Java250	Token-based	45,000	15,000	15,000
	Syntax-based	45,000	15,000	15,000
	Semantic-based	38,467	9,617	26,916
Python800	Token-based	144,000	48,000	48,000
	Syntax-based	144,000	48,000	48,000
	Semantic-based	151,652	37,914	50,434

In this study, we specifically evaluate the calibration of the model on the OOD datasets of Java250 and Python800. We simulate three OOD scenarios that along different data properties, namely token, syntax, and semantic, following established methodologies employed in prior studies[17] [14]. The statistics of the OOD datasets used in this work are listed at Table 4.

4.3.1 *OOD Datasets Creation.*

Token-based OOD. Considering that a code snippet is commonly regarded as a sequence of tokens and is directly fed to code models, the disparity in token sequences constitutes a fundamental distribution shift in code data. In the experiments, we utilize the token-based OOD datasets provided by Hu et al. [17]. To construct the datasets, frequency histograms are employed to derive density distributions of token sequences for each dataset. Subsequently, code snippets are grouped into two sets, namely ID and OOD data, by identifying tokens that are exclusively employed in specific sequences but remain unused in other sequences. The ID set is further randomly partitioned into training and validation sets.

Table 5: Experimental Results of Accuracy(%) and ECE (%) on Out-Of-Distribution Datasets of Java250

Dataset		ID	OOD-Token			OOD-Syntax			OOD-Semantic		
Models	Metrics	original	original	TS	LS	original	TS	LS	original	TS	LS
TextCNN	Acc	96.37	84.09	84.09	87.48	89.59	89.59	91.59	94.10	94.10	94.20
	ECE	0.38	2.84	2.99	24.01	1.40	3.40	20.28	2.48	5.64	22.36
ASTNN	Acc	96.52	80.21	80.21	82.94	88.59	88.59	91.64	91.57	91.57	92.44
	ECE	1.76	1.62	8.82	26.07	4.55	11.02	26.58	7.81	14.42	27.43
CodeBERTa	Acc	97.59	91.56	91.56	91.85	93.68	93.68	94.09	95.86	95.86	96.30
	ECE	0.90	3.68	0.60	6.33	2.75	0.54	5.28	1.01	3.50	7.28
CodeBERT	Acc	98.15	93.50	93.50	93.87	94.57	94.57	94.89	97.79	97.79	97.94
	ECE	0.87	3.43	1.24	5.75	2.65	0.97	6.31	1.01	1.40	6.28
GraphCodeBERT	Acc	98.29	93.99	93.99	94.05	94.92	94.92	94.97	97.95	97.95	98.07
	ECE	0.41	2.27	2.48	6.08	1.90	2.16	5.81	1.77	11.32	8.31
CodeT5	Acc	98.09	93.14	93.14	93.87	94.58	94.58	94.93	97.63	97.63	97.70
	ECE	0.61	2.86	1.34	8.05	1.53	0.61	7.59	1.07	0.49	8.61
UniXcoder	Acc	98.21	94.21	94.21	94.51	95.00	95.00	95.19	97.52	97.52	97.93
	ECE	0.75	1.77	2.08	5.01	1.65	1.66	4.76	0.58	6.86	5.49

Syntax-based OOD. Distinct from natural languages, programming languages inherently are more structured. Code models, such as ASTNN and GraphCodeBERT, incorporate structure information as input representations. The difference in structure constitutes syntactical distribution shift for code data. Similarly, we utilize the syntax-based OOD datasets provided by Hu et al. [17]. The average Robinson-Foulds distance between each code snippet and all other snippets is calculated by comparing their respective Concrete Syntax Trees(CSTs). Then code snippets with larger distances are classified as OOD sets, while examples with smaller distances are allocated to the training and validation sets.

Semantic-based OOD. In addition to the distribution shifts in terms of tokens and syntax, we also consider the semantic perspective to simulate the OOD scenario. Similar to the study [14], we utilize the pre-trained code model, CodeT5, to encode all samples in the dataset into 768-dimensional vectors. Subsequently, we employe Principal Component Analysis (PCA) to reduce the dimensionality of vectors to 50. We then utilize K-means algorithm to cluster the samples, with the value of K determined using the elbow method. For the Java250 dataset, we set K to 5, while for the Python800 dataset, we set K to 10. We randomly select 1/5 of the clusters as the OOD data, and the remaining data is divided into training and validation sets in a 4:1 ratio.

4.3.2 *OOD Calibration.*

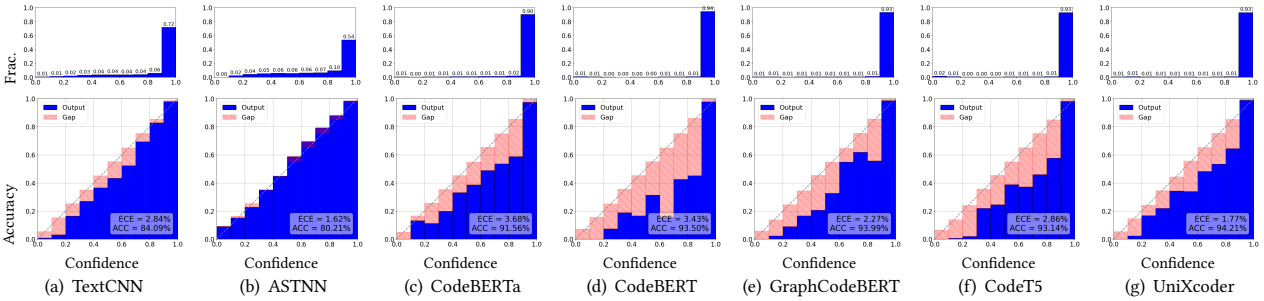
The experimental results of different models on OOD datasets of Java250 and Python800 are presented in Table 5 and Table 6, respectively. As shown in the table, the performance and calibration of code models have degraded in different OOD scenarios of Java250, particularly in the case of token-based and syntax-based OOD data. Similarly, the token-based OOD dataset of Python800 has a relatively larger impact on model performance, but the OOD datasets based on syntax and semantic show similar or even better performance compared to the ID dataset.

Several noteworthy observations can be drawn from the results. Firstly, pre-trained code models demonstrate higher stability and robustness in OOD settings compared to non-pre-trained models. Notably, pre-trained code models are able to maintain higher accuracy while the accuracy of non-pre-trained models decreases significantly on OOD data, especially in the case of token-based OOD. Secondly, pre-trained code models still suffer from the issue of over-confidence in different OOD scenarios. This is evident from Fig.5, which presents the confidence distribution and reliability diagrams of different code models on token-based OOD dataset of Java250. The figures reveal a considerable gap between the models' confidence and average accuracy for pre-trained code models, indicating that they are over-confident in OOD scenarios. Furthermore, it is noteworthy to mention that pre-trained code models exhibit higher levels of over-confidence compared to the ID settings. Fig.6 illustrates an example of CodeT5, where the gap between the model's confidence and average accuracy becomes larger when facing OOD data. On the other hand, non-pre-trained code models tend to be less confident in OOD settings, which may lead to a degradation in calibration.

We also conduct an evaluation of the effectiveness of temperature scaling and label smoothing in the context of OOD settings, as presented in Table 5 and Table 6. From the table, it can be observed that temperature scaling is not always effective in improving the calibration OOD settings. For instance, temperature scaling actually leads to an significant increase in ECE values of TextCNN and GraphCodeBERT on the OOD datasets of Java250, indicating the calibration degrades. This may be attributed to the fact that the temperature T is trained on the validation (ID) set, and the learned

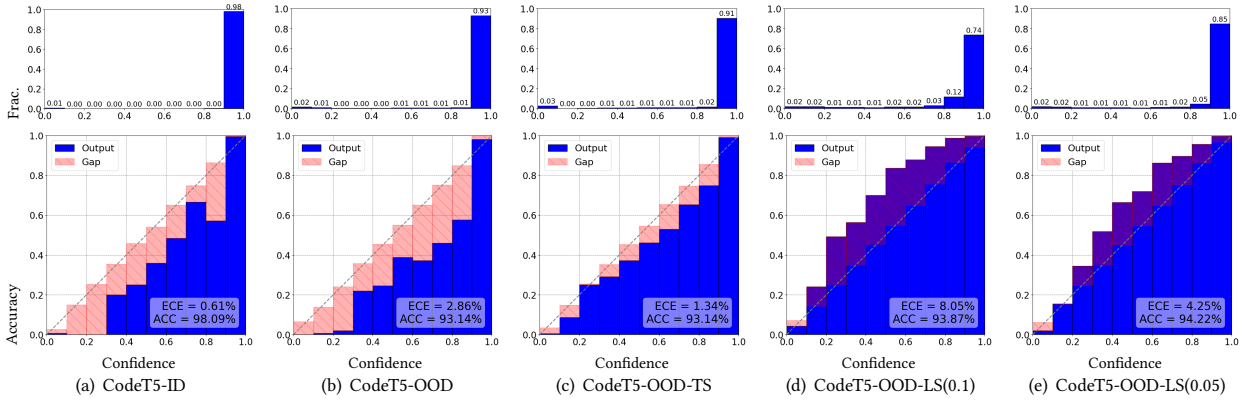
Table 6: Experimental Results of Accuracy(%) and ECE (%) on Out-Of-Distribution Datasets of Python800

Dataset		ID	OOD-Token			OOD-Syntax			OOD-Semantic		
Models	Metrics	original	original	TS	LS	original	TS	LS	original	TS	LS
TextCNN	Acc	97.71	90.94	90.94	92.37	97.48	97.48	97.93	96.88	96.88	97.29
	ECE	0.40	3.66	0.37	21.23	0.47	1.21	13.80	0.36	1.93	15.65
ASTNN	Acc	94.77	80.98	80.98	84.32	93.78	93.78	94.58	91.13	91.13	91.62
	ECE	0.76	1.37	6.51	25.43	1.75	6.57	17.75	1.33	6.90	19.38
CodeBERTa	Acc	98.38	93.83	93.83	94.54	98.38	98.38	98.58	98.15	98.15	98.29
	ECE	0.71	3.00	0.32	6.33	0.70	0.53	5.69	0.88	0.61	5.49
CodeBERT	Acc	98.64	94.48	94.48	95.16	98.79	98.79	98.83	98.59	98.59	98.73
	ECE	0.71	3.42	1.39	5.05	0.68	0.34	5.58	0.89	0.45	5.53
GraphCodeBERT	Acc	98.72	95.04	95.04	95.17	98.90	98.90	98.94	98.74	98.74	98.82
	ECE	0.48	2.30	1.16	6.14	0.44	1.54	5.91	0.61	1.51	5.64
CodeT5	Acc	98.44	95.15	95.15	95.68	98.74	98.74	98.90	98.72	98.72	98.88
	ECE	0.78	2.81	1.69	7.87	0.75	0.41	6.31	0.87	0.50	5.89
UniXcoder	Acc	98.72	95.46	95.46	95.67	98.88	98.88	98.94	98.61	98.61	98.72
	ECE	0.71	2.04	0.97	5.34	0.50	1.00	4.74	0.71	1.02	4.49



(a) TextCNN (b) ASTNN (c) CodeBERTa (d) CodeBERT (e) GraphCodeBERT (f) CodeT5 (g) UniXcoder

Figure 5: Confidence distribution (top row) and reliability diagrams (bottom row) for different code models on Functionality Classification (Java250-OOD-token).



(a) CodeT5-ID (b) CodeT5-OOD (c) CodeT5-OOD-TS (d) CodeT5-OOD-LS(0.1) (e) CodeT5-OOD-LS(0.05)

Figure 6: Confidence distribution (top row) and reliability diagrams (bottom row) for CodeT5 using Temperature Scaling(TS) and Label Smoothing(LS) on the Java250 OOD-token dataset. The CodeT5 model with TS exhibits good calibration, whereas LS may lead to over-calibration, resulting in potential under-confidence in predictions.

T may not be suitable to scale the logits of models in the test (OOD) set.

On the other hand, for models calibrated with label smoothing, we observe a slight improvement in accuracy under OOD scenarios, but at the same time, there is a noticeable increase in model ECE, indicating that label smoothing leads to a deterioration in calibration. Considering the fact that the models perform well due to the simplicity of the task, and the confidence of the models may be severely compromised due to over-calibration after applying label smoothing, as shown in Fig. 6. And we find a smaller value of label smoothing could mitigate this issue. For example, setting the value of label smoothing to 0.05 could result in better calibration compared to 0.1 (more details are available in our online materials). This suggests that an appropriate value of label smoothing should be carefully considered when calibrating models in OOD settings to avoid issues of over-calibration, particularly in the case of the model itself has small ECE values.

Summary. In OOD settings, the issue of over-confidence in pre-trained code models even worsens, while non-pre-trained models tend to exhibit lower confidence. And temperature scaling and label smoothing fail to adequately calibrate all code models.

5 THREATS TO VALIDITY

In this section, we analyze the potential threats to validity in our work, including the threats to the internal and external validity.

5.1 Internal Validity

In our experiments, the threats to internal validity primarily stem from the fine-tuning process of the code models, as their performance may vary in different environments and hyperparameter settings. To mitigate these threats, we employ the official implementations of various code models and fine-tune them on different tasks using the settings established in the studies conducted by Niu et al. [33] and Hu et al. [17]. The details of our experimental settings can be accessed in our online materials. The performance of our code models aligns with the results reported in these papers, indicating that our models are suitably trained.

5.2 External Validity

The threats to external validity primarily lie in the specific models, tasks, and calibration methods chosen in our experiments. We have focused on five well-known pre-trained code models. The calibration of other pre-trained code models requires further validation. Additionally, we conduct experiments on four code understanding tasks. It remains to be determined how to measure the model calibration on code generation tasks, duo to the inherent difficulties in accurately depicting the accuracy and probability in generation tasks. Future work should encompass the investigation of calibration in more pre-trained code models across a broader range of SE tasks. Furthermore, we have only investigated the effectiveness of temperature scaling and label smoothing as calibration methods for their representativity. There is a need for further research to advance the understanding and development of calibration techniques for pre-trained code models.

6 RELATED WORK

In this section, we briefly introduce the previous studies that are related to our work, including pre-trained code models and model calibration.

6.1 Pre-trained Code Models

In recent years, pre-trained code models have consistently achieved state-of-the-art performance levels in various code-related tasks [49] [33]. Notably, these pre-trained code models are typically based on the Transformer architecture [43]. And these models can be categorized into three types: encoder-only models, decoder-only models, and encoder-decoder models.

Encoder-only pre-trained code models utilize only the encoder part of the Transformer architecture, as the name suggests. There are several encoder-only models developed for programming languages, such as CodeBERT [9], GraphCodeBERT [12], CuBERT [19], and C-BERT [4]. These encoder-only models typically adopt the same architecture as BERT [8] and RoBERTa [26], and are commonly used to encode contextualized information of code for code understanding tasks. In contrast, decoder-only models utilize only the decoder part of the Transformer architecture. Examples of decoder-only code models include CodeGPT [27] and GPT-C [39], which are primarily designed for generative tasks such as code completion and code generation. There are also encoder-decoder models, such as PLBART [1], CodeT5 [44], and SPT-Code [34], which jointly train both encoder and decoder for comprehensive modeling of programming languages.

6.2 Model Calibration

Model calibration has been extensively studied in the fields of NLP and CV. Nguyen and O'Connor [31] proposed a posterior simulation method to analyze the calibration of probabilistic NLP models. Guo et al. [10] investigated the calibration of modern deep neural networks on image and document classification tasks, and they revealed that temperature scaling is effective for model calibration. Minderer et al. [28] revisited the calibration of modern deep neural networks, covering convolutional and non-convolutional architectures. Desai and Durrett [7], Chen et al. [5] explored calibration of pre-trained language models using techniques such as temperature scaling and label smoothing on both in-domain and out-of-domain datasets. Aiming to mitigate the miscalibration of pre-trained language models, Kong et al. [22] propose a regularization approach using pseudo samples. And Park and Caragea [35] improved model calibration using mixup. However, the study of model calibration for code models is still at a very early stage. And further research is needed to advance this area of study.

7 CONCLUSION

Model calibration is a crucial aspect of assessing the reliability of models, particularly in practical applications. In this study, we present a systematic investigation into the calibration of pre-trained models of source code, considering both in-distribution (ID) and

out-of-distribution (OOD) settings. Our experimental results reveal that pre-trained code models often suffer from the issue of overconfidence, while non-pre-trained models tend to exhibit lower confidence. And this disparity is more pronounced in OOD scenarios. Furthermore, we find that temperature scaling and label smoothing are effective in calibrating code models in ID data, but their effectiveness diminishes in OOD settings.

Several key insights emerge from our observations. Firstly, we emphasize the necessity of calibrating pre-trained code models before deploying them, especially in high-stakes scenarios. Secondly, while both calibration methods are effective in ID data, temperature scaling may be a more suitable choice due to its simplicity. When applying label smoothing, careful consideration of an appropriate smoothing value is crucial to avoid issues of over-calibration. Smaller values are generally preferred in simpler tasks. Furthermore, the results highlight the need for more effective calibration methods specifically tailored to pre-trained code models, especially in OOD settings.

These findings contribute to a deeper understanding of the calibration of pre-trained code models, and underscore the need for increased attention in improving their calibration. And the insights contribute to the reliable and robust deployment of pre-trained code models in various real-world applications.

REFERENCES

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 2655–2668. https://doi.org/10.18653/v1/2021.naacl-main.211

[2] Houssem Ben Braiek, Ali Tfaily, Foutse Khomh, Thomas Reid, and Ciro Guida. 2022. SmOOD: Smoothness-based Out-of-Distribution Detection Approach for Surrogate Neural Networks in Aircraft Design. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 94:1–94:13. https://doi.org/10.1145/3551349.3556936

[3] Glenn W. Brier. 1950. Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review* 78, 1 (Jan. 1950), 1. https://doi.org/10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2

[4] Luca Buratti, Saurabh Pujar, Mihaela A. Bornea, J. Scott McCarley, Yunhui Zheng, Gaetano Rossiello, Alessandro Morari, Jim Laredo, Veronika Thost, Yufan Zhuang, and Giacomo Domeniconi. 2020. Exploring Software Naturalness through Neural Language Models. *CoRR* abs/2006.12641 (2020). arXiv:2006.12641 https://arxiv.org/abs/2006.12641

[5] Yangyi Chen, Lifan Yuan, Ganqu Cui, Zhiyuan Liu, and Heng Ji. 2022. A Close Look into the Calibration of Pre-trained Language Models. *CoRR* abs/2211.00151 (2022). https://doi.org/10.48550/arXiv.2211.00151 arXiv:2211.00151

[6] Morris H. DeGroot and Stephen E. Fienberg. 1983. The Comparison and Evaluation of Forecasters. *Journal of the Royal Statistical Society. Series D (The Statistician)* 32, 1/2 (1983), 12–22. http://www.jstor.org/stable/2987588

[7] Shrey Desai and Greg Durrett. 2020. Calibration of Pre-trained Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 295–302. https://doi.org/10.18653/v1/2020.emnlp-main.21

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[9] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[10] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1321–1330. http://proceedings.mlr.press/v70/guo17a.html

[11] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 7212–7225. https://doi.org/10.18653/v1/2022.acl-long.499

[12] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=jLoC4ez43PZ

[13] Kartik Gupta, Amir Rahimi, Thalaiyasingam Ajanthan, Thomas Mensink, Cristian Sminchisescu, and Richard Hartley. 2021. Calibration of Neural Networks using Splines. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=eQe8DEWNN2W

[14] Hossein Hajipour, Ning Yu, Cristian-Alexandru Staicu, and Mario Fritz. 2022. SimSCOOD: Systematic Analysis of Out-of-Distribution Behavior of Source Code Models. *CoRR* abs/2210.04802 (2022). https://doi.org/10.48550/arXiv.2210.04802 arXiv:2210.04802

[15] Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. 2020. AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=S1gmrxHFvB

[16] Jens Henriksson, Christian Berger, Markus Borg, Lars Tornberg, Sankar Raman Sathyamoorthy, and Cristofer Englund. 2021. Performance analysis of out-of-distribution detection on trained neural networks. *Inf. Softw. Technol.* 130 (2021), 106409. https://doi.org/10.1016/j.infsof.2020.106409

[17] Qiang Hu, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Lei Ma, Mike Papadakis, and Yves Le Traon. 2022. CodeS: A Distribution Shift Benchmark Dataset for Source Code Learning. *CoRR* abs/2206.05480 (2022). https://doi.org/10.48550/arXiv.2206.05480 arXiv:2206.05480

[18] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and Evaluating Contextual Embedding of Source Code. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 5110–5121. http://proceedings.mlr.press/v119/kanade20a.html

[19] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and Evaluating Contextual Embedding of Source Code. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 5110–5121. http://proceedings.mlr.press/v119/kanade20a.html

[20] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). ACL, 1746–1751. https://doi.org/10.3115/v1/d14-1181

[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[22] Lingkai Kong, Haoming Jiang, Yuchen Zhuang, Jie Lyu, Tuo Zhao, and Chao Zhang. 2020. Calibrated Language Model Fine-Tuning for In- and Out-of-Distribution Data. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1326–1340. https://doi.org/10.18653/v1/2020.emnlp-main.102

[23] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 6402–6413. https://proceedings.neurips.cc/paper/2017/hash/9ef2ed4b7fd2c810847ffa5fa85bce38-Abstract.html

[24] Zenan Li, Xiaoxing Ma, Chang Xu, Jingwei Xu, Chun Cao, and Jian Lu. 2020. Operational calibration: debugging confidence errors for DNNs in the field. In

ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020. 901–913. https://doi.org/10.1145/3368089.3409696

[25] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel J. Orr, Lucia Zheng, Mert Yüksekgönül, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2022. Holistic Evaluation of Language Models. *CoRR* abs/2211.09110 (2022). https://doi.org/10.48550/arXiv.2211.09110 arXiv:2211.09110

[26] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[27] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, Joaquin Vanschoren and Sai-Kit Yeung (Eds.). https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c16a5320fa475530d9583c34fd356ef5-Abstract-round1.html

[28] Matthias Minderer, Josip Djolonga, Rob Romijnders, Frances Hubis, Xiaohua Zhai, Neil Houlsby, Dustin Tran, and Mario Lucic. 2021. Revisiting the Calibration of Modern Neural Networks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 15682–15694. https://proceedings.neurips.cc/paper/2021/hash/8420d359404024567b5aefda1231af24-Abstract.html

[29] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional Neural Networks over Tree Structures for Programming Language Processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, Dale Schuurmans and Michael P. Wellman (Eds.). AAAI Press, 1287–1293. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11775

[30] Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. 2015. Obtaining Well Calibrated Probabilities Using Bayesian Binning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, Blai Bonet and Sven Koenig (Eds.). AAAI Press, 2901–2907. http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9667

[31] Khanh Nguyen and Brendan O'Connor. 2015. Posterior calibration and exploratory analysis for natural language processing models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton (Eds.). The Association for Computational Linguistics, 1587–1598. https://doi.org/10.18653/v1/d15-1182

[32] Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005 (ACM International Conference Proceeding Series, Vol. 119)*, Luc De Raedt and Stefan Wrobel (Eds.). ACM, 625–632. https://doi.org/10.1145/1102351.1102430

[33] Changan Niu, Chuanyi Li, Vincent Ng, Dongxiao Chen, Jidong Ge, and Bin Luo. 2023. An Empirical Comparison of Pre-Trained Models of Source Code. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2136–2148. https://doi.org/10.1109/ICSE48619.2023.00180

[34] Changan Niu, Chuanyi Li, Vincent Ng, Jidong Ge, Liguo Huang, and Bin Luo. 2022. SPT-Code: Sequence-to-Sequence Pre-Training for Learning Source Code Representations. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 1–13. https://doi.org/10.1145/3510003.3510096

[35] Seoyeon Park and Cornelia Caragea. 2022. On the Calibration of Pre-trained Language Models using Mixup Guided by Area Under the Margin and Saliency. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 5364–5374. https://doi.org/10.18653/v1/2022.acl-long.368

[36] Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton. 2017. Regularizing Neural Networks by Penalizing Confident Output Distributions. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=HyhbYrGYe

[37] Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir R. Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. 2021. CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, Joaquin Vanschoren and Sai-Kit Yeung (Eds.). https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/a5bfc9e07964f8dddeb95fc584cd965d-Abstract-round2.html

[38] Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal Kumar Roy, and Mohammad Mamun Mia. 2014. Towards a Big Data Curated Benchmark of Inter-project Code Clones. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*. IEEE Computer Society, 476–480. https://doi.org/10.1109/ICSME.2014.77

[39] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. IntelliCode compose: code generation using transformer. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1433–1443. https://doi.org/10.1145/3368089.3417058

[40] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2818–2826. https://doi.org/10.1109/CVPR.2016.308

[41] Juozas Vaicenavicius, David Widmann, Carl R. Andersson, Fredrik Lindsten, Jacob Roll, and Thomas B. Schön. 2019. Evaluating model calibration in classification. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan (Proceedings of Machine Learning Research, Vol. 89)*, Kamalika Chaudhuri and Masashi Sugiyama (Eds.). PMLR, 3459–3467. http://proceedings.mlr.press/v89/vaicenavicius19a.html

[42] Ruslan Vasilev and Alexander D'yakonov. 2023. Calibration of Neural Networks. *CoRR* abs/2303.10761 (2023). https://doi.org/10.48550/arXiv.2303.10761 arXiv:2303.10761

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[44] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

[45] Yeming Wen, Ghassen Jerfel, Rafael Muller, Michael W. Dusenberry, Jasper Snoek, Balaji Lakshminarayanan, and Dustin Tran. 2021. Combining Ensembles and Data Augmentation Can Harm Your Calibration. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=g11CZSghXyY

[46] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *CoRR* abs/1910.03771 (2019). arXiv:1910.03771 http://arxiv.org/abs/1910.03771

[47] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural Attack for Pre-trained Models of Code. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 1482–1493. https://doi.org/10.1145/3510003.3510146

[48] Xi Ye and Greg Durrett. 2022. Can Explanations Be Useful for Calibrating Black Box Models?. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 6199–6212. https://doi.org/10.18653/v1/2022.acl-long.429

[49] Zhengran Zeng, Hanzhuo Tan, Haotian Zhang, Jing Li, Yuqun Zhang, and Lingming Zhang. 2022. An extensive study on pre-trained models for program understanding and generation. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea,*

July 18 - 22, 2022, Sukyoung Ryu and Yannis Smaragdakis (Eds.). ACM, 39–51. https://doi.org/10.1145/3533767.3534390

[50] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A novel neural source code representation based on abstract syntax tree. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 783–794. https://doi.org/10.1109/ICSE.2019.00086

[51] Shujian Zhang, Chengyue Gong, and Eunsol Choi. 2021. Knowing More About Questions Can Help: Improving Calibration in Question Answering. In *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021 (Findings of ACL, Vol. ACL/IJCNLP 2021)*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 1958–1970. https://doi.org/10.18653/v1/2021.findings-acl.172

[52] Yaqin Zhou, Shangqing Liu, Jing Kai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 10197–10207. https://proceedings.neurips.cc/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html