Unveiling Project-Specific Bias in Neural Code Models

Zhiming Li[1], **Yanzhou Li**[1], **Tianlin Li**[1,†], **Mengnan Du**[2], **Bozhi Wu**[1]
Yushi Cao[1], **Junzhe Jiang**[3], **Yang Liu**[1]
[1]Nanyang Technological University [2]New Jersey Institute of Technology
[3]Hong Kong Polytechnic University
{zhiming001, yanzhou001, tianlin001, bozhi001, yushi002}@e.ntu.edu.sg
mengnan.du@njit.edu, junzhe.jiang@connect.polyu.hk, yangliu@ntu.edu.sg

Abstract

Deep learning has introduced significant improvements in many software analysis tasks. Although the Large Language Models (LLMs) based neural code models demonstrate commendable performance when trained and tested within the intra-project independent and identically distributed (IID) setting, they often struggle to generalize effectively to real-world inter-project out-of-distribution (OOD) data. In this work, we show that this phenomenon is caused by the heavy reliance on project-specific shortcuts for prediction instead of ground-truth evidence. We propose a Cond-Idf measurement to interpret this behavior, which quantifies the relatedness of a token with a label and its project-specificness. The strong correlation between model behavior and the proposed measurement indicates that without proper regularization, models tend to leverage spurious statistical cues for prediction. Equipped with these observations, we propose a novel bias mitigation mechanism that regularizes the model's learning behavior by leveraging latent logic relations among samples. Experimental results on two representative program analysis tasks indicate that our mitigation framework can improve both inter-project OOD generalization and adversarial robustness, while not sacrificing accuracy on intra-project IID data. Our code is available at https://github.com/Lyz1213/BPR_code_bias.

Keywords: Bias Learning, Neural Code Models, Model Interpretation

1. Introduction

Neural network models have revolutionized the software engineering community by achieving significant improvements on many benchmarks, while not requiring much domain expert knowledge and manual efforts. The Transformer architecture-based Large Language Models (LLMs) are nowadays the most prevalent neural code models by demonstrated to be effective on many downstream tasks. Concretely, the encoder-only LLMs have achieved improvements in many program analysis tasks (e.g., bug detection, clone detection, etc. (Feng et al., 2020; Guo et al., 2021)), the encoder-decoder LLMs (Ahmad et al., 2021; Raffel et al., 2020; Wang et al., 2021), as well as the decoder-only LLMs (Chen et al., 2021; Touvron et al., 2023), are especially useful for sequential prediction tasks (e.g., code summarization (Al-Kaswan et al., 2023; Gu et al., 2022), etc.). In particular, for the encoder-only LLMs, the CodeBert (Feng et al., 2020) model and its variants (Guo et al., 2021, 2022) have surpassed many delicately designed model architectures without much inductive bias thanks to the power of the pretraining and fine-tuning (Devlin et al., 2019; Brown et al., 2020) training paradigm.

Despite the success reported in the literature, we notice that most of these neural code models are evaluated merely under the intra-project independent identically distributed (IID) data-split setting (Zhou et al., 2019; Allamanis et al., 2020), i.e., collect code samples (often at the function level) from multiple projects, then randomly shuffle and split the dataset for training and test. However, in real-world scenarios, neural code models should be trained and tested in the inter-project setting for the majority of cases, i.e., trained on samples from a fixed set of projects while tested on samples from previously unseen projects. It is obvious that the inter-project setting is much more challenging since the vocabulary within different projects varies considerably because the naming conventions among developers differ. Thus, the real-world inter-project evaluation setting could be considered as out-of-distribution (OOD), due to the significant amount of usage of out-of-vocabulary words (Bojanowski et al., 2017; Hu et al., 2019) and different programming styles. In particular, for the representative program analysis tasks we evaluated in this paper, neural code models that achieve decent performance on the intra-project data suffer from a significant performance drop on the inter-project data. In addition, previous empirical analysis indicates that neural code models are also sensitive to semantic-preserving adversarial attacks (Li et al., 2020) such as variable renaming and dead code insertion (Yefet et al., 2020; Bielik and Vechev, 2020).

In this work, we aim to explore the reason why the encoder-only LLMs-based neural code models have low generalization ability on inter-project data and why they are vulnerable to naive adversarial attacks. Toward this end, we probe the model

† Corresponding author

LREC-COLING 2024, pages 17205–17216
20-25 May, 2024. © 2024 ELRA Language Resource Association: CC BY-NC 4.0

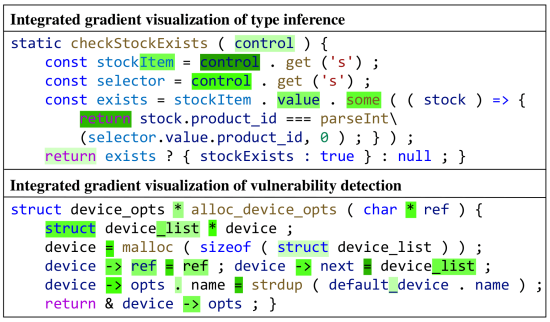

Figure 1: Illustrative examples of attribution vectors in terms of integrated gradient for cases of type inference and vulnerability detection. The shades of green indicate the weight value of the respective token in the attribution vector.

behavior with a DNN model explanation algorithm: integrated gradient (Sundararajan et al., 2017), and find that the models heavily rely on ungeneralizable project-specific cues for prediction while ignoring ground-truth evidence when trained without regularization. We formulate it as project-specific bias learning behavior. As shown in the examples of type inference (a task that aims to infer type for variables of an optionally-typed language) and vulnerability detection (a task that aims to predict whether a code snippet contains vulnerability) in Figure 1 (see Section 3.1 for detailed descriptions of the two tasks). To predict the type of variable `stockExists`, the GraphCodeBERT (GCB) model relies primarily on uninformative sub-words such as `control`, `Item` from irrelevant variable names in the snippet. In contrast, human developers would infer based on the Boolean constant `true` in the declaration statement and infer it as a Boolean. Similarly, for the case of vulnerability detection, to predict whether the snippet contains a vulnerability, the model distributes almost all its weights to the user-defined function/variable names, while ignoring the relevant memory management API `malloc`. The vulnerability can be easily identified based on the fact that the `malloc` function is not used along with a memory deallocation operation. This learning behavior is problematic when applying the model under the inter-project setting or adversarial setting, since the semantics of these user-defined variable/function names are inconsistent across projects. Furthermore, we show that project-specific bias learning behavior can be interpreted with a proposed measurement termed as conditional inverse document frequency (Cond-Idf), which measures the relatedness of a token with a label and its project specificness. We observe that when trained without regularization, the model would rely heavily on the tokens that frequently co-occur with a label yet are highly semantically inconsistent and ungeneralizable for prediction.

Furthermore, for the prevalent bias mitigation methods we evaluated in this work, we observe that though these methods manage to mitigate the model from using observed shortcuts[1], there is no guarantee that the post-mitigated model would infer based on the expected behavior instead of resorting to other unexpected bias. To handle this concern, we propose a novel bias mitigation mechanism, termed as BPR (Batch Partition Regularization). The proposed regularization is based on the principle of invariant risk minimization (IRM) (Arjovsky et al., 2019), which explicitly regularizes the model behavior by identifying common logic properties among samples. Concisely, the BPR first unshuffles and sorts samples in the training dataset according to a measure that embeds prior knowledge about logic relations. The training dataset is then divided into batches of *environments* in which the samples are the most closely correlated. The in-batch representations are expected to be regularized during gradient update such that logically correlated samples would share similar representations instead of embedding other unknown shortcuts after debiasing the known ones. The major contributions of our work are summarized as follows:

- We unveil that the previous state-of-the-art encoder-only LLMs-based neural code models trained under the intra-project setting would suffer from considerable performance drop on real-world inter-project/adversarial data, and show that this phenomenon can be attributed to the project-specific bias learning behavior.
- We indicate that the project-specific bias learning behavior can be interpreted with a proposed measurement called Cond-Idf.
- We propose a novel shortcut mitigation method, called BPR. The idea is to explicitly regularize the model behavior during training by identifying common logic properties among samples.
- Experimental results on two representative program analysis tasks validate that BPR can effectively improve inter-project OOD generalization and adversarial robustness while not sacrificing accuracy on intra-project IID data.

2. Methodology

In this section, we first introduce the analysis and interpretation methods of the project-specific bias learning behavior. Then we illustrate the details of the proposed bias mitigation mechanism, called batch partition regularization (BPR).

[1]we use bias and shortcut interchangeably in this paper.

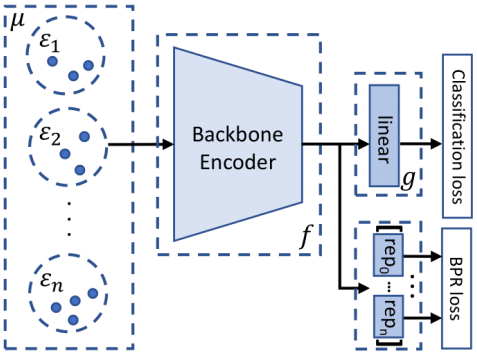

Figure 2: Overview of the proposed bias mitigation method. Training samples $x \in \mathcal{X}$ are first embedded with μ and sorted in terms of the similarity measure κ, then batchified and partitioned into multiple *environments* $\varepsilon \in \mathcal{E}$ according to their labels and similarity scores. Finally, the batch partition regularization (BPR) loss is computed along with the classification loss.

Algorithm 1: BPR mitigation mechanism

Data: Training set $\mathcal{X}$, similarity function κ, embedding function μ, encoder f

```
// calculate similarity matrix
```
$\mathcal{X}_\mu \leftarrow ()$;
for $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$ **do**
  $\lambda_{ij} \leftarrow \kappa(\mu(x_i), \mu(x_j)), \lambda \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$

```
// training set unshuffling
```
for $\lambda_{ij} \in sorted(vec(\lambda \backslash \{\lambda_{ij} | i = j\}))$ **do**
  for $k \in \{x_i, x_j\}$ **do**
    if $k \notin \mathcal{X}_\mu$ **then**
      $\mathcal{X}_\mu \leftarrow \mathcal{X}_\mu || k$;

```
// train with BPR loss
```
for $B \in batches(\mathcal{X}_\mu)$ **do**
  $\mathcal{L}_{\text{BPR}} \leftarrow \mathbb{E}_{(x_i,x_j)\sim B\times B} \mathbb{1}^{ij} \kappa(\mu(x_i), \mu(x_j)) \cdot$
    $\mathrm{S_c}(f(x_i;\theta), f(x_j;\theta))$
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{DEBIAS}} + \gamma_p \cdot \mathcal{L}_{\text{BPR}}$
  `// update model weights`
  $\theta \leftarrow \theta - \eta \nabla_\Theta \mathcal{L}$

2.1. Behavior Analysis and Interpretation

We consider two typical software analysis tasks: vulnerability detection and type inference (see Section 3.1 for a detailed description of the two tasks). The backbone encoder-only LLMs-based models we analyze are the pre-trained CodeBERT (Feng et al., 2020) and GraphCodeBERT (Guo et al., 2021, 2022) models. Although they have been reported to achieve state-of-the-art performance on multiple software analysis tasks (Guo et al., 2021, 2022), we find that compared to their high performance on the intra-project IID data, their performance drops considerably on the real-world inter-project OOD set and adversarial set for the two evaluated tasks. Intuitively, to robustly infer, the model should learn to embed abstract, generalizable code semantics, instead of using merely low-level self-defined variable/function names. In the following, we unveil that neural code models may utilize ungeneralizable tokens as shortcuts, while disregarding ground-truth evidence for predictions. This is because these tokens co-occur frequently with the label while they are project-specific and ungeneralizable due to the developer's idiosyncrasies, which results in models' poor generalization and robustness.

Model Behavior Analysis. We analyze the model's behavior using a post-hoc DNN model explanation algorithm: Integrated Gradient (Sundararajan et al., 2017). Intuitively, the algorithm attributes the prediction to the input by giving each feature an importance score (Montavon et al., 2018), which indicates its contribution to the output. The detailed formula is as follows:

$$\mathrm{IGs}(x_i) = (x_i - x_i') \cdot \frac{1}{m} \sum_{k=1}^{m} \frac{\partial f_y\left(x_i' + \frac{k}{m}(x_i - x_i')\right)}{\partial x_i} \tag{1}$$

Specifically, given an input sequence with T words $x_i = \{x_i^t\}_{t=1}^T$, $x_i^t \in \mathbb{R}^d$ denotes a word embedding with d dimensions, the model $f(\cdot)$ outputs the prediction probability $f_y(x_i)$ for the ground truth label y, m denotes the number of intermediate samples over the straightline path from baseline reference vector x_i' to the input x_i. We then compute the gradients of $f_y(x_i)$ with respect to each input word embedding within x_i and reduce each vector of the gradients to a single attribution value with the L2 norm. We use zero word embedding as the baseline reference vector x_i'. Eventually, we obtain a feature importance vector $\mathrm{IGs}(x_i) \in \mathbb{R}^T$, where each scalar within the vector indicates the contribution of the corresponding word to $f_y(x_i)$.

Skewed Dataset Distribution. To reflect the project-specific bias learning behavior, we propose a measurement called conditional inverse document frequency (Cond-Idf), which measures the co-occurrence between a word w and a label l ($\texttt{co-occur}(w, l)$), as well as its specificness across projects Π ($\texttt{specific}(w, \Pi)$). The measurement is denoted as follows:

$$\begin{aligned} \text{Cond-Idf}\ (w, l, \Pi) &= \texttt{co-occur}(w, l) \wedge \texttt{specific}(w, \Pi) \\ &= p(l \mid w) \cdot \mathrm{Idf}(w, \Pi) \\ &= p(l \mid w) \cdot \log \frac{N}{|\{\pi \in \Pi : w \in \pi\}|} \end{aligned} \tag{2}$$

We approximate the conjunction with product t-norm (Esteva and Godo, 2001). Specifically, the

conditional probability is calculated as $p(l|w) = \frac{count(w,l)}{count(w)}$, and N is the total number of projects in the corpus: $N = |\Pi|$. We normalize the Idf term such that both the two measurements are scaled between $[0, 1]$. For each label l, we obtain a distribution in terms of all words in the vocabulary. Words with high Cond-Idf values indicate that they frequently co-occur with the label and are also highly project-specific. We observe that a large portion of this part of the words is the user-defined components. Although these words strongly correlate with the label in the training set, their semantics are inconsistent and ungeneralizable. For example, consider a case in type inference, a model might correlate a token `temp` with the semantics of a Boolean object in one project, since it is frequently used in cases such as `temp=True;`. However, if the model learns this spurious correlation, it might arbitrarily infer an integer variable `temp` as Boolean when dealing with `temp=1.0;` since it bases its prediction heavily on token `temp` while ignoring the ground truth evidence `1.0`.

To interpret the bias learning behavior quantitatively, we evaluate the alignment between the model's integrated gradient distribution and its corresponding Cond-Idf distribution. We first calculate the integrated gradient importance vector for every sample in the IID test set. Afterward, we calculate the mean integrated gradient value for every token in the test set vocabulary and sort them in descending order. We then use polynomial regression to approximate its corresponding Cond-Idf distribution and measure its correlation with the integrated gradient distribution.

2.2. Proposed Mitigation Mechanism

Although many prevalent bias mitigation baselines manage to improve generalization and robustness by removing known bias, there is no guarantee that the debiased model would infer based on the expected behavior of developers instead of resorting to other unknown biases (Yoo and Qi, 2021). Motivated by this concern, we propose a novel mitigation mechanism called batch partition regularization (BPR) to regularize the behavior of neural code models (see Figure 2). BPR follows the *invariant-risk-minimization* (IRM) (Arjovsky et al., 2019) philosophy and aims to constrain neural code representation so that the model's learning behavior is expected to be invariant when handling samples with similar syntactic and semantic evidence. Details of the BPR algorithm are shown in Algorithm 1.

Dataset Unshuffling. Given a training set $\mathcal{X}$, we first compute a similarity matrix $\lambda \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$ of the training set (see line 1-3). Specifically, for samples $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$, we first map them to a measure space with the embedding function μ, then calculate their similarity with the similarity function κ: $\lambda_{ij} \leftarrow \kappa(\mu(x_i), \mu(x_j))$. Intuitively, this measures the level of logic closeness (similarity) of syntactic or semantic invariance between samples. For type inference, we focus on variables with assignment, we use bag-of-words (BoW) (Harris, 1954) vector that consists of tokens within the assignment statement of the target word as embedding μ and use cosine similarity $\mathrm{S_c}$ as κ to calculate logic closeness: $\kappa(\mu(\cdot), \mu(\cdot)) = \mathrm{S_c}(BoW(ASSIGN(\cdot)), BoW(ASSIGN(\cdot)))$, where $ASSIGN(\cdot)$ denotes the function that extracts the body of the assignment statement of the variable. The intuition is that the body of the assignment statement often contains consistent and generalizable information that is related to the type of the variable. For vulnerability detection, we use an encoder model pre-trained only on adversarial samples as μ (we use CodeBERT in this work). The adversarial samples are constructed from semantic-preserving identifier name replacement, denoted as ρ (refer to Section 3.1). The intuition is that this encoder would only rely on the generalizable code semantics for detection as the biased user-defined token names are all normalized. The similarity score between a sample pair (x_i, x_j) is calculated as: $\lambda_{ij} = \mathrm{S_c}(\epsilon(\rho(x_i)), \epsilon(\rho(x_j)))$, where $\epsilon(\cdot)$ denotes the pre-trained embedding model, $\mathrm{S_c}$ denotes cosine similarity function. The training samples are then unshuffled and sorted according to the vectorized similarity matrix $sorted(vec(\lambda \backslash \{\lambda_{ij} | i = j\}))$ (see line 4-7). In this way, after batchifying, each mini-batch $B \in batches(\mathcal{X}_\mu)$ would consist of samples with the closest logic relations.

In-batch Regularization. Finally, during training, within each mini-batch, we use the cosine embedding loss $\mathbb{1}^{ij}\mathrm{S_c}(f(x_i;\theta), f(x_j;\theta))$ to regularize the model into using similar representation when embedding samples with the same label and close logic relations. Here, f denotes the backbone encoder, $\mathbb{1}^{ij}$ is a boolean operator that selects samples with the same label. We weigh the loss of each sample pair with their corresponding similarity score λ_{ij} to prevent misalignment. Detailed BPR loss is as follows:

$$\mathcal{L}_{\mathrm{BPR}} = \mathbb{E}_{(x_i,x_j)\sim B\times B}\mathbb{1}^{ij}\kappa(\mu(x_i), \mu(x_j)) \cdot \mathrm{S_c}(f(x_i;\theta), f(x_j;\theta)). \quad (3)$$

$\mathcal{L}_{\mathrm{BPR}}$ can be trained together with existing mitigation methods and our final loss function is: $\mathcal{L} = \mathcal{L}_{\mathrm{DEBIAS}} + \gamma \cdot \mathcal{L}_{\mathrm{BPR}}$. In this work, we combine BPR with adversarial training/gradient reversal methods. $\mathcal{L}_{\mathrm{DEBIAS}}$ denotes loss function for adversarial training (Yefet et al., 2020)/gradient reversal (Stacey

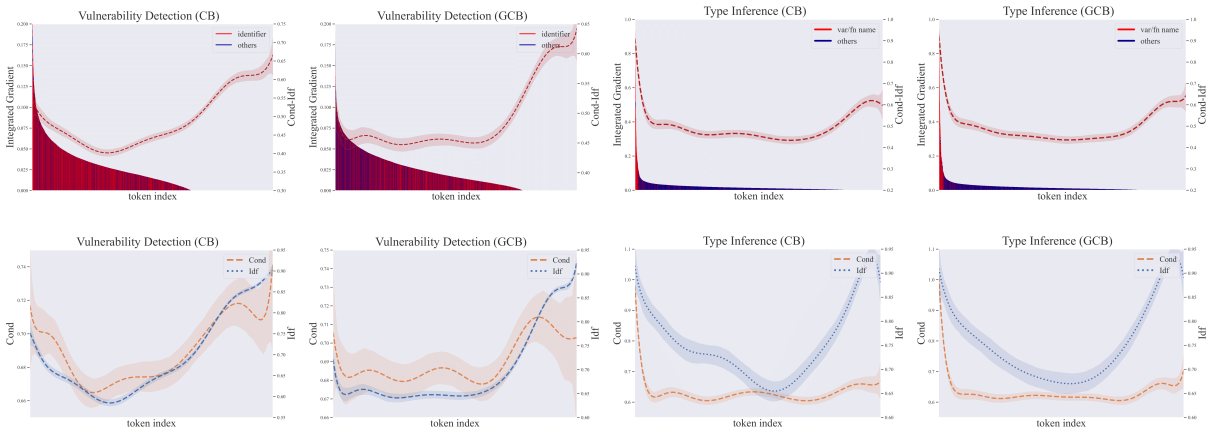

Figure 3: For the figures in the first row: the lower distribution in each figure is the ranked integrated gradient distribution. The upper red dashed line denotes the polynomial regression approximation of the corresponding Cond-Idf distribution in terms of token index. CB and GCB denote the CodeBERT and GraphCodeBERT models. For the figures in the second row: The blue and orange lines denote the polynomial regression approximation of the Cond and Idf distribution in terms of token index respectively.

	CodeBERT			GraphCodeBERT		
#Words	Top1	Top2	Top3	Top1	Top2	Top3
Ratio	66.6%	83.0%	90.0%	13.2%	21.4%	28.8%

Table 1: The ratio of samples whose top-n ($n \in \{1, 2, 3\}$) interpretation words contain user-defined identifiers (vulnerability detection).

	CodeBERT			GraphCodeBERT		
#Words	Top1	Top2	Top3	Top1	Top2	Top3
Ratio	25.6%	31.6%	34.9%	27.5%	33.0%	36.9%

Table 2: The ratio of samples whose top-n ($n \in \{1, 2, 3\}$) interpretation words contain user-defined identifiers (type inference).

et al., 2020) (see Section 3.1). The hyperparameter γ denotes the regulatory coefficient for BPR loss.

3. Experiments

3.1. Experimental Setup

Tasks & Datasets. We experiment with two representative program analysis tasks:

- *Type Inference*: The goal of this task is to predict the type for variables/parameters/functions in a code snippet written in optionally typed language. In this work, we use the TypeScript dataset (Hellendoorn et al., 2018). After preprocessing, the dataset contains samples from 233 TypeScript projects. We follow previous work (Hellendoorn et al., 2018) and split the dataset by project into 80%, 10% and 10% for inter-project OOD/adversarial training validation and test set. The training portion is also randomly shuffled and split by 80-20 proportions for IID train and test. We perform a semantic-preserving non-targeted attack (Yefet et al., 2020) by replacing variable/parameters/function names with a set of dummy variables e.g., {`var_0`, `var_1`, ...} on the corresponding inter-project samples to form the adversarial set.
- *Vulnerability Detection*: Previous work formulates vulnerability detection as a sequence/graph classification task, in which given a code snippet, the neural model should learn to predict whether it contains vulnerability or not. In this work, we collect 999 C-language open-source projects from GitHub that contain vulnerabilities via keyword filtering in the commit message. We split the dataset by project into 70% for IID training and testing (randomly split by 80-20 proportions), 10% and 20% for inter-project OOD/adversarial validation and test set.We use the same adversarial set construction as the type inference task.

Mitigation Baselines. We evaluate BPR along with four representative mitigation baselines: two model-agnostic methods (reweighting, product-of-expert (PoE)) and two representation-based methods (adversarial training, gradient reversal). For reweighting (Schuster et al., 2019; Clark et al., 2019), it first obtains a biased model by training the model only on the biased features, then the output probability of the bias-only model p_b is used to adjust the weights of the training samples to train the debiased model, such that the contribution of samples to which the biased model assigns high prediction probability is lower weighted. For the PoE (He et al., 2019; Clark et al., 2019; Mahabadi et al., 2020), it also requires a trained biased model,

Methods	CodeBERT						GraphCodeBERT					
	Top-1 Acc (%)			Top-5 Acc (%)			Top-1 Acc (%)			Top-5 Acc (%)		
	INTRA	INTER	ADV	INTRA	INTER	ADV	INTRA	INTER	ADV	INTRA	INTER	ADV
Original	94.82	82.91	66.01	98.35	94.87	86.00	94.54	83.02	65.32	98.77	95.05	84.53
Reweighting	94.19	83.22	66.81	98.73	95.65	86.24	94.74	83.63	66.24	98.97	95.88	84.76
PoE	91.66	80.24	64.50	96.74	89.74	82.21	93.84	82.56	64.73	97.62	90.29	82.48
Grad-rev	94.34	83.16	66.26	98.32	95.28	85.73	94.62	83.58	66.01	98.69	95.63	84.38
Grad-rev w/ BPR	94.90	83.95	66.51	98.85	95.84	86.77	95.37	84.42	66.20	98.93	96.06	84.82
Adv-train	94.92	83.42	79.55	98.84	96.08	94.27	95.45	83.74	79.32	99.08	95.82	94.06
Adv-train w/ BPR	95.25	84.45	80.57	98.87	96.24	94.31	95.68	84.95	80.44	99.12	96.08	95.06

Table 3: Generalization and robustness evaluation on the type inference task.

the debiased model is trained by ensembling its output probability with that of the biased model. For adversarial training (Madry et al., 2017; Yefet et al., 2020), it optimizes the model based on both original samples and perturbed adversarial samples so that the co-occurrence of spurious data cues and labels are down-weighted. Finally, for gradient reversal (Stacey et al., 2020; Kim et al., 2019; Minervini and Riedel, 2018), it unlearns the bias in a minimax game by predicting the target bias using model representation and reverses its gradient during backpropagation.

Implementation Details. We focus on the analysis of two encoder-only LLMs-based neural code models: the pretrained CodeBERT (CB) and GraphCodeBERT (GCB) models. We use the open-sourced checkpoints from Feng et al. (Feng et al., 2020) and Guo et al. (Guo et al., 2021). For both benchmarks, we fine-tune the model for 10 epochs, which all models could converge. We use the Adam optimizer for the update and the learning rate is set as $2 * 10^{-5}$. The training batch size is set as 16. We conducted all experiments on a Ubuntu 18.04 server with 24 cores of 2.20GHz CPU, 251GB RAM and two Quadro RTX 8000 GPUs.

3.2. Project-Specific Bias Analysis & Interpretation

In this section, we quantitatively analyze and interpret the project-specific bias learning behavior of the CB and GCB models.

Bias Behavior Analysis. We calculate the mean integrated gradient for each token in the IID test set vocabulary and rank them in descending order to obtain the sorted distribution. Then, for vulnerability detection, we perform lexical analysis and categorize the vocabulary into user-defined identifiers (denoted as identifier) and others. The identifier category contains user-defined variable/function names and macro-definition names. As shown in Figure 3 (first row), we illustrate the integrated gradient weights of the project-specific tokens with red bars and others with blue bars. For type inference, we categorize tokens into user-defined Boolean variable/function name identifiers and others. Similarly, we use red and blue bars to represent the integrated gradient weights of these two categories, as shown in Figure 3 (first row). As shown in the distribution, we observe that the area under the head of the distribution is heavily reddish for both models in terms of the type inference task, which indicates that the models focus heavily on user-defined components. Quantitatively, we take the top 1% of the tokens as the head. While the user-defined identifiers only occupy 15.0% of the vocabulary, 88.1% of tokens within the head fall into this category for the CB model, and 83.6% for the GCB model. The results indicate that the data-flow aware pretraining objectives of the GCB model allow it to be less biased than the CB model on the type inference task which is reliant on explicit def-use relations. The result is consistent for the vulnerability detection task. With the top 1% of the tokens as the head, while the self-defined tokens take up 56.6% of the vocabulary, 66.8% of the tokens within the head belong to the user-defined components for the CB model, while it drops to 39.7% for the GCB model. Furthermore, we measure the extent of the bias learning behavior from the sample level. Specifically, we calculate the ratio of samples whose top-n ($n \in \{1, 2, 3\}$) integrated gradient tokens contain user-defined identifiers. The results are shown in Table 1,2. The results indicate that the behavior of both the CB and GCB models are biased for a considerable amount of samples. For example, for vulnerability detection (CB), up to 90.0% of the samples leverage project-specific user-defined tokens as its top-3 attribution words for prediction.

Bias Behavior Interpretation. Given the ranked integrated gradient distribution, we calculate the corresponding Cond-Idf distribution in terms of the token index and approximate it with polynomial regression (order=10). As shown in the first row of Figure 3, we observe that for the head part of the distribution where models give relatively high attribution weights, the fitted curve of Cond-Idf is positively correlated with the integrated gradient distribution for both tasks. We further conduct a detailed ablation analysis of the Cond-Idf measurement, the results are shown in the second row of

Methods	CodeBERT						GraphCodeBERT					
	Top-1 Acc (%)			F1-score (%)			Top-1 Acc (%)			F1-score (%)		
	INTRA	INTER	ADV	INTRA	INTER	ADV	INTRA	INTER	ADV	INTRA	INTER	ADV
Original	81.61	64.01	61.50	82.37	67.94	67.83	81.70	64.48	64.34	83.53	68.51	68.18
Reweighting	80.34	61.99	58.64	81.06	67.35	67.31	80.67	63.78	59.93	81.85	67.49	66.2
PoE	81.24	63.09	58.12	81.90	68.02	67.73	81.28	63.61	61.37	83.04	68.39	67.25
Grad-rev	81.66	64.52	63.21	83.05	69.00	68.82	81.27	64.17	64.07	82.78	69.86	69.78
Grad-rev w/ BPR	81.73	64.57	64.56	83.20	69.21	69.38	81.57	65.12	65.01	82.33	70.05	70.02
Adv-train	81.89	64.53	63.31	82.60	68.92	68.80	81.91	64.95	64.98	84.07	68.89	68.89
Adv-train w/ BPR	82.76	64.72	64.83	83.24	69.37	69.41	82.46	66.31	66.38	84.47	70.18	70.18

Table 4: Generalization and robustness evaluation on the vulnerability detection task.

Figure 3. Specifically, we break the measurement into the conditional probability distribution approximation (Cond) and Idf distribution approximation. For the two evaluated models on both the vulnerability detection and type inference tasks, we can observe that the head of the ranked integrated gradient distribution positively correlates with both the Cond and Idf distributions. Quantitatively, to calculate the correlation, we sample 50 data points that are evenly spaced from the top 50% of both the ranked integrated gradient distribution and the Cond/Idf distributions. We use the Spearman's rank correlation as the measurement. *E.g.*, for the GCB model on the type inference task, the head of the Cond measurement and the ranked integrated gradient distribution are correlated with a Spearman's rank correlation of $\rho_{\text{Cond}} = 0.566$; and for the Idf measurement: $\rho_{\text{Idf}} = 1.00$. Similarly, for the CB model on the vulnerability detection task, the two measurements correlate with the ranked integrated gradient distribution with high rank correlation ($\rho_{\text{Cond}} = 0.561$, $\rho_{\text{Idf}} = 0.700$).

3.3. Bias Mitigation Effectiveness

We present results on the intra-project IID test set, inter-project OOD set and adversarial set. In Table 3, we evaluate the type inference performance in terms of top-1 and top-5 accuracy following previous works (Wei et al., 2020; Jesse et al., 2021). In Table 4, we evaluate the vulnerability detection performance in terms of top-1 accuracy and F1-score. We have the following three key findings:

- With the standard IID training-test split, both models achieve decent performance on the intra-project IID test set relying on shortcut features, whereas their performance drops significantly on the inter-project OOD and adversarial set. For example, for type inference, CB achieves 94.82%@top1-Acc on the intra-project test set and drops to 82.91% on the inter-project OOD and further decreases to 66.01% on adversarial data. The results indicate that the bias learning behavior seriously undermines models' generalizability and robustness.

- Among the baselines evaluated, model-agnostic mitigation approaches (reweighting and PoE) are less helpful compared to representation-based methods (adversarial training and gradient reversal). For example, for vulnerability detection (CB), adversarial training increases adversarial robustness accuracy by +1.81%@top1-Acc, while reweighting and PoE decrease it by -2.86% and -3.38% respectively. The results are similar for type inference. One possible reason is that there is no clear boundary for defining biased and unbiased sample, since every code sample requires the usage of user-defined words like variable/ function names, thus representation-based mitigation methods are more effective compared with model-agnostic methods.

- When combined with Grad-rev and Adv-train, BPR demonstrates a consistent capacity to enhance both generalization and robustness by learning more robust representation. Notably, it is intriguing to observe that the implementation of BPR can even yield enhancements in terms of IID performance. For example, for vulnerability detection (GCB), BPR improves adversarial training by +1.40%@top1-Acc, +1.29%@F1 on the adversarial set. To better understand the source of improvement, we compute the mean integrated gradient values of the ground-truth tokens that the model should focus on for the original model, model trained with adversarial training and model trained with adversarial training w/ BPR (we experiment on the GCB model since it achieves overall better performance). For vulnerability detection, we compute the mean IG value of the memory management API of the C standard library. And for type inference, we compute the value of the boolean constant and logical operators. The results are shown in Figure 4. Specifically, adversarial training significantly improves over the original GCB model, and by incorporating BPR, the mean IG values of the ground-truth tokens are all further increased, which indicates that the model's behavior is much more robust and focuses more on the ground-truth tokens. We further conduct a case study of vulnerability

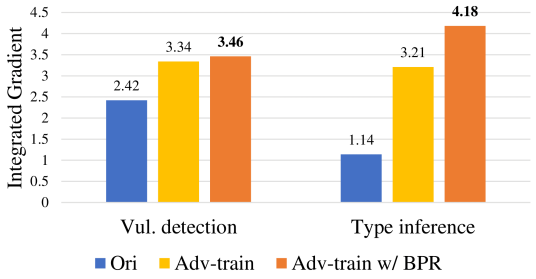

Figure 4: Mean integrated gradient of the original and mitigated GCB models on the ground-truth evidence.

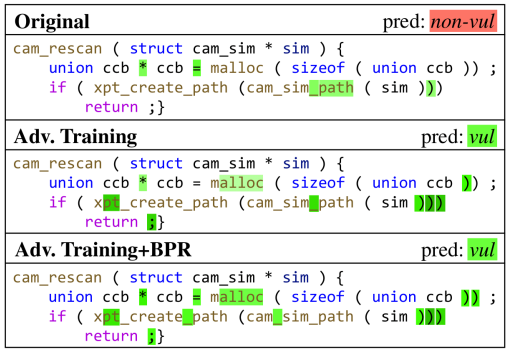

Figure 5: Illustrated example of applying BPR compared to the baseline methods for the vulnerability detection task.

detection as shown in Figure 5. The union `ccb` is first allocated with memory using the C standard API `malloc`. However, it would cause a memory leak as the program directly return while failing to free up the memory properly. It is obvious that the original model completely ignores the ground-truth API token `malloc` and erroneously predicts the function as non-vulnerable. When trained with adversarial training, model starts focusing on it and infers correctly. Finally, when incorporated with BPR, model robustly predicts it as vulnerable by paying much higher attention to the ground-truth evidence.

4. Related Work

Shortcut Learning & Mitigation. DNNs have been shown powerful and prevalent in many areas (He et al., 2015; Vaswani et al., 2017; Hu et al., 2021, 2023; Huang et al., 2023). However, researchers observe that neural models tend to leverage shallow statistical cues instead of generalizable features for prediction (Du et al., 2023; Li et al., 2023a,b). For the VQA task, it is found that the model often conditions its predictions on language prior while ignoring the image (Manjunatha et al., 2019a; Agrawal et al., 2018). For the NLI task, models tend to focus on a single branch of input or frequent but spurious unigram words (Schuster et al., 2019; Niven and Kao, 2019). One of the most representative line of mitigation method is debiasing from bias-only model, which includes re-weighting (Schuster et al., 2019), product-of-expert (He et al., 2019; Clark et al., 2019; Zhou and Bansal, 2020; Cadene et al., 2019), etc. Representation-based methods are another line of effort that is proven to be effective. The idea is to orthogonalize model's representation from the bias features (Madry et al., 2017; Stacey et al., 2020; Kim et al., 2019).

Code Representation Learning. Neural models are useful in modeling programming language data and perform well in software analysis tasks, e.g., vulnerability detection (Zhou et al., 2019), program synthesis (Chen et al., 2018), etc.

The Transformer architecture-based Large Language Models (LLMs) have become the most widely used neural code models due to their state-of-the-art performance on a wide range of downstream tasks. These models can be categorized into three major groups according to their architectures (Hou et al., 2023): encoder-only (Feng et al., 2020; Guo et al., 2021), encoder-decoder (Ahmad et al., 2021; Raffel et al., 2020; Wang et al., 2021), and decoder-only (Chen et al., 2021; Touvron et al., 2023) models. Many of the previous neural code models literature conduct evaluation only under the *intra-project* setting instead of the *inter-project* setting, despite the fact that the latter is closer to reality. In addition, previous work notices that these neural code models are vulnerable to naive adversarial attack (Yefet et al., 2020) such as variable name change or dead code insertion.

5. Conclusion and Future Work

In this work, we analyze the project-specific bias learning behavior of the encoder-only LLMs-based neural code models, which renders them ungeneralizable to inter-project OOD or adversarial settings. We observe that this phenomenon can be interpreted via the Cond-Idf measurement. Furthermore, we propose a general mitigation mechanism BPR that forces the model to infer based on robust representation using logic relations among samples. Experimental results on two representative benchmarks validate that BPR improves OOD generalization and adversarial robustness while not sacrificing IID performance. In the future, we plan to study bias learning behavior on more benchmarks and model architectures.

6. Acknowledgments

This research is supported by the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN) and NRF Investigatorship NRF-NRFI06-2020-0001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

Limitations

Despite that BPR can effectively increase the generalization and robustness of the model, one limitation of our approach is that it requires careful design of the modeling of human expert knowledge. However, for many software analysis tasks, human expert knowledge is suboptimal or difficult to abstract. Thus automatic identification of latent bias for neural code model would be an important future direction.

Ethics Statement

We state that the vulnerability detection dataset we used in this work is collected from GitHub via keyword mapping in the commit message and has been through rigorous human review, in which all vulnerabilities are fully disclosed and repaired by developers, and shall contain no sensitive information or exposure of privacy. Thus, it would not produce any potential negative societal consequences.

7. Bibliographical References

Antonio A Abello, Roberto Hirata, and Zhangyang Wang. 2021. Dissecting the high-frequency bias in convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 863–871.

Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. 2018. Don't just assume; look and answer: Overcoming priors for visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4971–4980.

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. *arXiv preprint arXiv:2103.06333*.

Ali Al-Kaswan, Toufique Ahmed, Maliheh Izadi, Anand Ashok Sawant, Premkumar Devanbu, and Arie van Deursen. 2023. Extending source code pre-trained language models to summarise decompiled binarie. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 260–271. IEEE.

Miltiadis Allamanis, Earl T Barr, Soline Ducousso, and Zheng Gao. 2020. Typilus: Neural type hints. In *Proceedings of the 41st acm sigplan conference on programming language design and implementation*, pages 91–105.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. 2019. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Pavol Bielik and Martin Vechev. 2020. Adversarial robustness for code. In *International Conference on Machine Learning*, pages 896–907. PMLR.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Remi Cadene, Corentin Dancette, Matthieu Cord, Devi Parikh, et al. 2019. Rubi: Reducing unimodal biases for visual question answering. *Advances in neural information processing systems*, 32:841–852.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Xinyun Chen, Chang Liu, and Dawn Song. 2018. Execution-guided neural program synthesis. In *International Conference on Learning Representations*.

Christopher Clark, Mark Yatskar, and Luke Zettlemoyer. 2019. Don't take the easy way out: Ensemble based methods for avoiding known dataset biases. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Mengnan Du, Fengxiang He, Na Zou, Dacheng Tao, and Xia Hu. 2023. Shortcut learning of large language models in natural language understanding.

Mengnan Du, Varun Manjunatha, Rajiv Jain, Ruchi Deshpande, Franck Dernoncourt, Jiuxiang Gu, Tong Sun, and Xia Hu. 2021. Towards interpreting and mitigating shortcut learning behavior of nlu models. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Francesc Esteva and Lluıs Godo. 2001. Monoidal t-norm based logic: towards a logic for left-continuous t-norms. *Fuzzy sets and systems*, 124(3):271–288.

Richard Evans and Edward Grefenstette. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547.

Robert W Floyd. 1962. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. 2019. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *Seventh International Conference on Learning Representations*.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Jian Gu, Pasquale Salza, and Harald C Gall. 2022. Assemble foundation models for automatic code summarization. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 935–946. IEEE.

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*, pages 7212–7225. Association for Computational Linguistics.

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.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

He He, Sheng Zha, and Haohan Wang. 2019. Unlearn dataset bias in natural language inference by fitting the residual. *arXiv preprint arXiv:1908.10763*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Vincent J Hellendoorn, Christian Bird, Earl T Barr, and Miltiadis Allamanis. 2018. Deep learning type inference. In *Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 152–162.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language

models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620*.

Ming Hu, Jiepin Ding, Min Zhang, Frédéric Mallet, and Mingsong Chen. 2021. Enumeration and deduction driven co-synthesis of ccsl specifications using reinforcement learning. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 227–239. IEEE.

Ming Hu, Zeke Xia, Dengke Yan, Zhihao Yue, Jun Xia, Yihao Huang, Yang Liu, and Mingsong Chen. 2023. Gitfl: Uncertainty-aware real-time asynchronous federated learning using version control. In *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE.

Ziniu Hu, Ting Chen, Kai-Wei Chang, and Yizhou Sun. 2019. Few-shot representation learning for out-of-vocabulary words. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4102–4112.

Yihao Huang, Qing Guo, and Felix Juefei-Xu. 2023. Zero-day backdoor attack against text-to-image diffusion models via personalization. *arXiv preprint arXiv:2305.10701*.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50.

Kevin Jesse, Premkumar T Devanbu, and Toufique Ahmed. 2021. Learning type annotation: is big data enough? In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1483–1486.

Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, and Junmo Kim. 2019. Learning not to learn: Training deep neural networks with biased data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9012–9020.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. Bert-attack: Adversarial attack against bert using bert. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6193–6202.

Tianlin Li, Qing Guo, Aishan Liu, Mengnan Du, Zhiming Li, and Yang Liu. 2023a. Fairer: Fairness as decision rationale alignment.

Tianlin Li, Cao Yue, Zhang Jian, Zhao Shiqian, Huang Yihao, Liu Aishan, Guo Qing, and Liu Yang. 2023b. Runner: Responsible unfair neuron repair for enhancing deep neural network fairness. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 66–78. IEEE Computer Society.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Rabeeh Karimi Mahabadi, Yonatan Belinkov, and James Henderson. 2020. End-to-end bias mitigation by modelling biases in corpora. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8706–8716.

Varun Manjunatha, Nirat Saini, and Larry S Davis. 2019a. Explicit bias discovery in visual question answering models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9562–9571.

Varun Manjunatha, Nirat Saini, and Larry S Davis. 2019b. Explicit bias discovery in visual question answering models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9562–9571.

Pasquale Minervini and Sebastian Riedel. 2018. Adversarially regularising neural nli models to integrate logical background knowledge. *Proceedings of the 22nd Conference on Computational Natural Language Learning (CoNLL 2018)*.

Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. 2018. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15.

Timothy Niven and Hung-Yu Kao. 2019. Probing neural network comprehension of natural language arguments. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Judea Pearl et al. 2000. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 19.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Tal Schuster, Darsh J Shah, Yun Jie Serene Yeo, Daniel Filizzola, Enrico Santus, and Regina Barzilay. 2019. Towards debiasing fact verification models. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Joe Stacey, Pasquale Minervini, Haim Dubossarsky, Sebastian Riedel, and Tim Rocktäschel. 2020. Avoiding the hypothesis-only bias in natural language inference via ensemble adversarial training. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328. PMLR.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Haohan Wang, Zexue He, Zachary C Lipton, and Eric P Xing. 2018. Learning robust representations by projecting superficial statistics out. In *International Conference on Learning Representations*.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*.

Jiayi Wei, Maruth Goyal, Greg Durrett, and Isil Dillig. 2020. Lambdanet: Probabilistic type inference using graph neural networks. *The International Conference on Learning Representations (ICLR)*.

Noam Yefet, Uri Alon, and Eran Yahav. 2020. Adversarial examples for models of code. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–30.

Jin Yong Yoo and Yanjun Qi. 2021. Towards improving adversarial training of nlp models. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 945–956.

Jerrold H Zar. 2005. Spearman rank correlation. *Encyclopedia of biostatistics*, 7.

Xiang Zhou and Mohit Bansal. 2020. Towards robustifying nli models against lexical dataset biases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8759–8771.

Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems*, 32.

Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. 2021. A robustly optimized bert pre-training approach with post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pages 1218–1227.