Dynamic Data Fault Localization for Deep Neural Networks

Yining Yin
ynyin@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yang Feng*
fengyang@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Shihao Weng
shweng@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Zixi Liu
zxliu@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yuan Yao
y.yao@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yichi Zhang
201250088@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Zhihong Zhao
zhaozhih@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Zhenyu Chen*
zychen@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

ABSTRACT

Rich datasets have empowered various deep learning (DL) applications, leading to remarkable success in many fields. However, data faults hidden in the datasets could result in DL applications behaving unpredictably and even cause massive monetary and life losses. To alleviate this problem, in this paper, we propose a **d**ynamic data **fau**lt **lo**calization approach, namely DFauLo, to locate the mislabeled and noisy data in the deep learning datasets. DFauLo is inspired by the conventional mutation-based code fault localization, but utilizes the differences between *DNN mutants* to amplify and identify the potential data faults. Specifically, it first generates multiple DNN model mutants of the original trained model. Then it extracts features from these mutants and maps them into a suspiciousness score indicating the probability of the given data being a data fault. Moreover, DFauLo is the first *dynamic* data fault localization technique, prioritizing the suspected data based on user feedback, and providing the generalizability to unseen data faults during training. To validate DFauLo, we extensively evaluate it on 26 cases with various fault types, data types, and model structures. We also evaluate DFauLo on three widely-used benchmark datasets. The results show that DFauLo outperforms the state-of-the-art techniques in almost all cases and locates hundreds of different types of real data faults in benchmark datasets.

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*.

KEYWORDS

fault localization, deep learning testing, data quality

ACM Reference Format:
Yining Yin, Yang Feng, Shihao Weng, Zixi Liu, Yuan Yao, Yichi Zhang, Zhihong Zhao, and Zhenyu Chen. 2023. Dynamic Data Fault Localization for Deep Neural Networks. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616345

1 INTRODUCTION

Deep learning (DL)-based systems have achieved tremendous success in a wide range of applications such as facial recognition [15], autonomous driving [30, 88], medical diagnosis [55], etc. Compared to traditional code-driven software, DL-based systems adopt a data-driven programming paradigm, and thus face extra security and reliability challenges related to *data faults* (e.g., mislabeled data samples). Such faults can be inherited by DL models that are trained on the data, resulting in significant losses and even fatalities [23, 53, 71]. Therefore, guaranteeing the quality of datasets becomes a significant quality assurance aspect for DL-based systems.

Related Work. Various quality assurance techniques specifically designed for DNN models have been proposed, including diagnosing incorrect training code and hyperparameter settings [81], locating misbehavior inside the model neurons [25], generating

*Yang Feng and Zhenyu Chen are the corresponding authors.

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3616345

adversarial inputs [59, 68], and repairing the DL model misbehavior [76, 77]. The above techniques mainly aim at finding the vulnerability in a well-trained model, but neglect the root cause of such vulnerability, i.e., the faults in the dataset. Such faults may propagate to the models in an implicit way (through the model's internal parameters), and fundamentally influence the performance and quality of DL applications [71].

In the machine learning community, learning with noisy labels (LNL) has been proposed to handle the potential noises in the dataset [74]. These works aim to train a robust and accurate model even with the presence of label noises. However, LNL techniques fail to debug the dataset and improve the data quality, and thus cannot guarantee the performance of downstream DL applications [53].

According to a recent survey conducted by IBM [23], 96% of enterprises encounter data challenges in their AI projects, and 2/5 of them are not confident in ensuring data quality. Recently, a few research groups [53, 65, 71] have also conducted empirical studies and advocated focusing on the data aspect rather than the model aspect when developing DL applications. For example, Liang et al. [53] reveal that the design and sculpting of the data used to develop AI often rely on bespoke manual work, which critically affects the trustworthiness of the model. Sambasivan et al. [71] reported that a large number of experienced developers are impacted by at least one data cascade-compounding event, which leads to negative downstream effects from data issues. Curtis et al. [8] have found systematic labeling errors (ranging from 0.15% to 10.12%) in popular AI benchmark datasets, which destabilizes the evaluation of DL benchmarks [65]. In a nutshell, both industry and academia are calling for an effective and efficient data-centric approach to improve the data quality for DNNs.

Our Work. In this paper, we introduce DFauLo, a **d**ynamic data **fau**lt **lo**calization approach for deep neural networks. The key insight of DFauLo is to incorporate the manual inspection feedback into the inspection process, and thus prioritize the suspicious data adaptively. While existing techniques [64, 96] are designed to prioritize the data into a static queue for manual inspection, DFauLo adopts a different method that dynamically updates the data queue based on the feedback of each round of inspections. DFauLo migrates the classic mutation-based code fault localization techniques [50, 62, 66, 67] to obtain effective data fault features. By generating multiple DNN model mutants from different dimensions of the original model, we propose and extract fault features based on the prediction behavior of each mutant. Then DFauLo models the suspiciousness model with extracted features, and dynamically updates the model with collected feedback.

Specifically, the key insight of DFauLo is two-fold. First, DFauLo amplifies and captures the difference between clean data and sparsely distributed faulty data in the dataset via *model mutation*. In the classic mutation-based code fault localization, mutating the program statement could amplify the execution difference of the test suite. That is, faulty statements tend to perform differently compared to correct statements when two groups of mutants are generated, one that mutates a faulty statement and the other that mutates a correct statement. To apply this idea to data fault localization, we introduce a set of model mutation strategies to fine-tune the original DNN model into multiple *slightly* changed mutated models. These model mutation strategies ensure that DNN mutant has a similar prediction capability on clean data compared to the original model, but may not well fit the sparse faulty data. In other words, the mutated model amplifies the difference between clean and faulty data.

Second, DFauLo spots the potential faulty data via an iterative way, where the feedback obtained from the manual review is utilized to *dynamically* enhance its performance and generalizability to unseen data faults during the training. Specifically, since data distributions change in practice and it is impossible to collect all types of data faults during training, we propose a dynamic data fault localization workflow that can adapt to the real-world distribution of data faults. Our approach employs a logistic regression model, called *Susp*, to predict the suspicious faulty data. We then use previously confirmed faulty data to continuously update the *Susp*, allowing the DFauLo to dynamically prioritize suspected data and improve the effectiveness of subsequent data fault localization.

To validate the effectiveness of DFauLo, we conduct a comprehensive experiment on 26 subjects of different data fault types, model structures, data types, and DL tasks. The experimental results show that DFauLo can effectively locate different types of data faults, and the model performance can be improved by correcting the located faults. Additionally, we also conduct a case study on three widely-used benchmark datasets, including EMNIST-Letter [22], ImageNet [24], and MTFL [4]. Compared with baseline methods, DFauLo locates more real faults in the EMNIST-Letter dataset. And for more complicated benchmark datasets, hundreds of real faults of different types are confirmed by DFauLo.

Contributions. The contributions of this paper include:

- **Technique.** We propose DFauLo, a data fault localization technique for dynamically locating data faults in DL datasets. To the best of our knowledge, DFauLo is the first technique that incorporates the model mutation method to locate data faults in a dynamic manner. Since it is designed upon features extracted from general attributes of DNNs, DFauLo is applicable for various types of deep learning datasets.
- **Tool.** We have implemented DFauLo into a tool that can automatically locate data faults hidden in the deep learning datasets. We have also released the source code and presented a demonstration at [12].
- **Study.** We have conducted a large-scale experiment with 26 subjects containing both text and image data to validate the effectiveness of DFauLo. Further, we apply DFauLo to three widely-used deep learning benchmark datasets and locate hundreds of real data faults. This result further confirms the practicability of DFauLo.

2 PRELIMINARY AND BACKGROUND

This section introduces the background knowledge of deep learning and states the definitions of data fault and data fault localization.

2.1 Deep Learning

2.1.1 Model Training. A deep neural network (DNN) model can be denoted as $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{Y}$, which is essentially a mapping from the input domain $\mathcal{X}$ to the output domain $\mathcal{Y}$. Take image classification as an example. The input $x \in \mathcal{X}$ is an image, and the output $y \in \mathcal{Y}$ is a one-hot vector representing the image's label over the label space. Typically, training a DNN model often requires a large dataset, denoted as $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. Given a

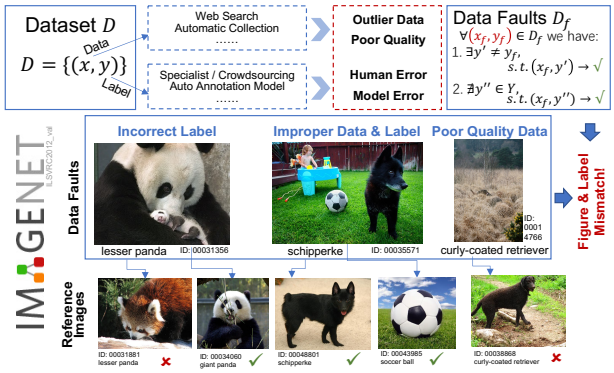


Figure 1: Detected data fault samples in the ImageNet dataset.

loss function $\mathcal{L}(\mathcal{F}(x), y)$ such as cross-entropy [20] and an optimization strategy such as stochastic gradient descent (SGD) [75], the model parameters can be updated based on the training set D. After sufficient epochs of training on D, we can obtain proper internal parameters of the DNN model and deploy it in the application to predict labels for unseen inputs. Essentially, just as the code faults explicitly influence the program behavior through test case execution [62, 83], defects hidden in the dataset impact the model's internal parameters during training, which in turn leads to potential vulnerabilities in the model [53, 71].

2.1.2 Data Preparation. Data preparation is an integral part of deep learning, and it includes collecting, cleaning, and labeling the data samples. Among them, data labeling is the most labor-intensive. Benefiting from online crowdsourcing platforms such as Amazon Mechanical Turk (AMT) [1], Appen (previously known as CrowdFlower) [3], and Prolific [10], the developers or data engineers of DL applications can hire a large number of crowd-workers to manually label the raw-data. Although such online platforms provide diverse solutions [2, 9, 44] to help data engineers adjust their data labeling tasks, it is still inevitable to include various errors, biases, and noises into the dataset [37]. Therefore, developers or data engineers of DL applications are required to ensure the data quality and thus improve the generalizability and reliability of DL models trained on the dataset [53].

2.2 Problem Statement

2.2.1 Data Fault. Data faults refer to the mismatches between data and labels in the dataset, which can occur due to poor data quality [28], or mistakes made by human organizers [78] or annotators [61]. Ideally, the data instance in the dataset should associate with and only with one label; however, in practice, due to various noise or mistakes, some data might associate with no or more than one label. In that situation, these data may be faulty and could threaten the performance of deep learning models. Thus, to clarify the problem this paper is dealing with, we assume each data is associated with a label set. We define data faults as follows:

Definition 1 (**Data Fault**). *Let $\mathcal{X}$ denote the data instance set and $\mathcal{Y}$ denote the label set. Suppose $x \in \mathcal{X}$ is an input data instance in a manually labeled dataset $D = \{(x, y) | x \in \mathcal{X}, y \in \mathcal{Y}\}$, we can denote the manual label of x as y and its ground truth label set as Y_x. Based on this definition, we say $(x, \hat{y})$ is a correctly labeled data, if and only if Y_x contains the only label $\hat{y}$. Therefore, in the manually constructed dataset D containing faults, we say (x_f, y_f) is a **data fault**, if: 1) $\exists y' \neq y_f$, $y' \in Y_x$; or 2) $Y_x = \emptyset$. Moreover, if there exists the only $y' \in \mathcal{Y}$, such that (x_f, y') is not a data fault, we name the corresponding fault type as **label noise**; otherwise, the fault is categorized as **data noise**.*

Figure 1 presents some examples of data faults detected in the image dataset ImageNet [24]. The first example ($image_1$, *lesser panda*) is classified into the incorrect class, which can be corrected by modifying the class label. We name this type of data fault as **label noise** [74, 86], which assumes the data sample is labeled into the incorrect classes. The second example ($image_1$, *schipperke*) does not include all appropriate labels. To correct such data, the input should be cropped or modified before re-annotating. The third example ($image_3$, *curlycoated retriever*) has poor quality, and we cannot infer any label from its input. To correct such data, we should delete it from the dataset. We name the fault type of the second and third examples as **data noise**, because correcting such data fault requires modifying the input data. [1]

2.2.2 Data Fault Localization. In classic software engineering research, fault localization techniques aim to yield a ranking list of program statements according to their suspiciousness. A higher suspiciousness score indicates the statement is more likely to contain faults, i.e., the buggy code that causes failures [50, 67]. Inspired by the classic fault localization problem, we migrate the concept of fault localization into the DNN dataset debugging, and propose to assist data engineers by ranking the data samples so that data faults can be prioritized for review and correction. We next define the data fault localization problem:

Definition 2 (**Data Fault Localization Problem**). *Let $D_f \subset D$ be a subset of the dataset D, which contains all data faults of D. The data fault localization problem is to find a suspiciousness function $Susp : D \rightarrow \mathbb{R}$ such that $\forall (x_f, y_f) \in D_f$ and $\forall (x, y) \in D \backslash D_f$, we have:*

$$Susp((x_f, y_f)) > Susp((x, y)).$$

3 APPROACH

This section presents the proposed approach, DFauLo. We first present the overall idea and then detail its three steps.

3.1 The Key Idea

Our approach is built upon the insight that the identified data faults represent the distribution of potential faults in the dataset, and utilizing inspection feedback can improve the fault localization capability. Therefore, designing a dynamic data fault localization technology that introduces feedback information into subsequent data inspection is a natural idea. To locate data faults dynamically, we need to model a suspiciousness function that can be updated with feedback. By extracting the fault features of each data point, we can dynamically fit a suspiciousness model *Susp* based on the fault features. Therefore, we have to design a fault feature extraction method that can reflect the differences between clean and faulty data. Considering that DNN models initially learn and fit the primary features of the dataset [17, 38, 87], models fine-tuned on the majority subset are expected to fluctuate on data faults that are

[1] This type of data is also known as *outlier* [42] or *out-of-distribution* [33] data.

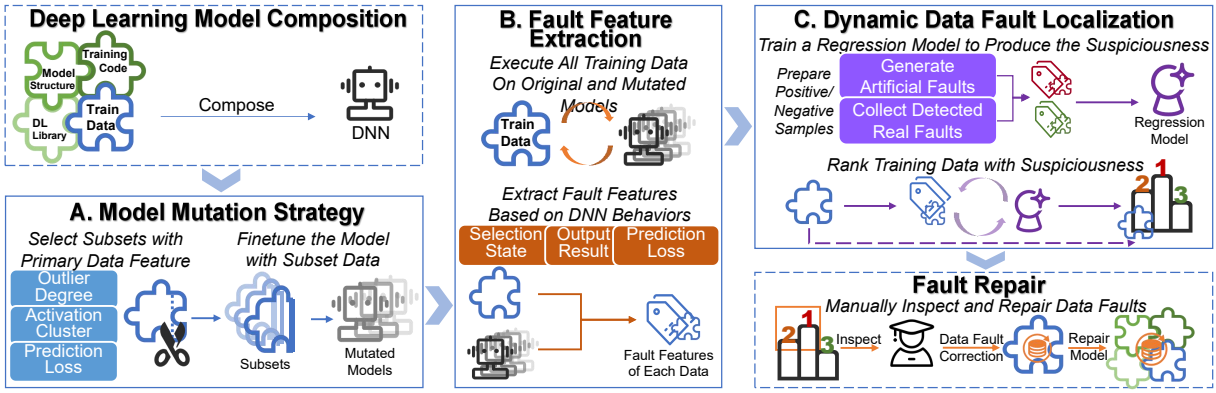


Figure 2: An overview of the data fault localization approach DFauLo.

sparsely distributed and account for a small proportion of the set. Based on this insight, we introduce the concept of model mutation to amplify the behavior differences between clean data and data faults. Specifically, DFauLo first fine-tunes the original model to generate multiple model mutants, then designs and extracts fault features for each data point based on the behavior divergence of model mutants. With distinctive fault features and a dynamic iteration process design, DFauLo can update the suspiciousness model *Susp* based on feedback to locate diverse data faults in application.

The overview of our approach DFauLo is shown in figure 2, which consists of three key steps. First, to amplify and capture the divergence between clean data and data faults in the training dataset D, the primary features of the data from different model layers are sampled to generate model mutants, which is described in detail in Section 3.2. Second, each data sample in D is predicted by the model mutants, and their corresponding features related to the selection state, the output result, and the prediction loss are extracted as data fault features. The feature extraction method is presented in Section 3.3. Finally, based on the extracted features, a dynamic data fault location approach is proposed. A logistic regression model *Susp* is applied to map the fault features into suspiciousness scores for each data sample. To improve the localization precision, we propose to update the *Susp* with feedback gathered from previously checked data, which is detailed and expressed in Section 3.4.

3.2 Model Mutation Strategy

Locating faults via mutation has been discussed in conventional fault localization apporaches [62, 66]. If the faulty program statement is mutated, failed test cases will pass on mutated programs; if the correct statement is mutated, previously passed test cases may fail. Therefore, the mutation strategy could amplify the behavior between clean components and faults. Migrating such an idea into deep learning, if the original model is fine-tuned and the prediction capability is *almost* unchanged, the clean data will be predicted correctly as the original model, and the fault data may be predicted differently if the model mutant is not over-fitted to them.

The remaining problem is how to fine-tune the model so as to maintain an *almost* unchanged model mutant. To this end, we design various mutation strategies that could keep the majority of the dataset to mutate the original model, so that the model mutant is slightly different from the original one and outputs similar predictions to samples with typical features.

To implement this design, we sample the subset D' based on the neuron outputs across the input, hidden, and output layers of the original DNN model $\mathcal{F}$. Because a DNN model contains multiple hidden layers and deeper layer outputs are considered to provide more abstract information [26], we mainly employ the last linear layer to represent to hidden layer's information in this paper. For each layer, we identify a small ratio subset of data with sparse features, which are denoted as D^1_{out}, D^2_{out}, and D^3_{out}, and discard them to obtain $D^i = D \backslash D^i_{out}$. After that, the model mutants are generated by retraining the original model $\mathcal{F}$ for several extra epochs on D^i to slightly change the original model, which is lightweight than train the model from scratch. Denoting the predefined remove ratio as α, we next present the details to obtain each D^i_{out}.

3.2.1 ***Input Layer.*** For the input layer, we capture the data features by detecting input outliers from the input space. Specifically, we adopt a classical method, namely VAE, i.e., variational auto-encoder [40], as the outlier detector. It builds an auto-encoder to extract the low-dimensional features of each data. If an input $x \in X$ has a larger reconstruction error, it is supposed to have more suspicious data features and be an outlier. Denote the reconstruction error of x as $E(x)$. The data is sorted according to the value of $E(x)$, and the prioritized α of the largest errors is discarded. The subset to be removed is formally denoted as:

$$D^1_{out} = \{(x, y) | (x, y) \in D \wedge \underset{|X'|/|X|=\alpha}{\arg\max} \ (\Sigma_{x \in X'} E(x))\}.$$

The remaining samples $D^1 = D \backslash D^1_{out}$ is then applied for train the model mutant $\mathcal{F}^1$.

3.2.2 ***Hidden Layer.*** The hidden neurons of the model carry the features learned from the training data, and the deeper layer of the network represents more abstract information about model behaviors [19]. In the training dataset, neuron activation states of the clean data corresponding to the same label are similar, while a small amount of noise data may trigger different neuron activation [21]. Therefore, we choose to remove the data with deviations in the distribution of neuron activation states. For each input data $x \in X$, we collect the activation vector of the last linear layer for analysis, which is denoted as $\mathcal{F}_{-1}(x)$. Then, we adopt k-Means [20] to cluster the activation states $AS = \{\mathcal{F}_{-1}(x) | x \in X\}$ into two clusters AS_1, AS_2. Obviously, the sparse features would belong to a minor cluster, thus for each dataset, we remove the data in the smaller cluster to construct the subset. Suppose $|AS_1| \geq |AS_2|$, the

Table 1: Features extracted for each (x, y) in dataset D.

Type	Formalization	Description
SS	$\mathbb{1}_{D^1}((x,y))$	Data (x,y) belongs to D^1
	$\mathbb{1}_{D^2}((x,y))$	Data (x,y) belongs to D^2
	$\mathbb{1}_{D^3}((x,y))$	Data (x,y) belongs to D^3
OR	y	Original label y of input x
	$\mathcal{F}(x)$	Output result of input x on model $\mathcal{F}$
	$\mathcal{F}^1(x)$	Output result of input x on model $\mathcal{F}^1$
	$\mathcal{F}^2(x)$	Output result of input x on model $\mathcal{F}^2$
	$\mathcal{F}^3(x)$	Output result of input x on model $\mathcal{F}^3$
PL	$\mathcal{L}(\mathcal{F}(x), y)$	Prediction loss of data (x,y) on model $\mathcal{F}$
	$\mathcal{L}(\mathcal{F}^1(x), y)$	Prediction loss of data (x,y) on model $\mathcal{F}^1$
	$\mathcal{L}(\mathcal{F}^2(x), y)$	Prediction loss of data (x,y) on model $\mathcal{F}^2$
	$\mathcal{L}(\mathcal{F}^3(x), y)$	Prediction loss of data (x,y) on model $\mathcal{F}^3$

subset to be removed is formally defined as:

$$D^2_{out} = \{(x,y) | (x,y) \in D \wedge \mathcal{F}_{-1}(x) \in AS_2\}.$$

Similar to the input layer mutation, we apply $D^2 = D \backslash D^2_{out}$ to train the second model mutant $\mathcal{F}^2$.

3.2.3 ***Output Layer.*** We analyze the model from the output perspective based on prediction loss. If the DNN fails to fit the data due to unrecognizable or mismatched labels, the corresponding prediction loss will be large. Hence, based on the output layer, we choose to remove the data with high prediction losses. Denoting the loss function of DNN as $\mathcal{L}(\mathcal{F}(x), y)$, the removed data is formalized as:

$$D^3_{out} = \{(x,y) | (x,y) \in D \wedge \underset{|X'|/|X|=\alpha}{\arg\max} \ (\Sigma_{x \in X'} \mathcal{L}(\mathcal{F}(x), y))\}.$$

The subset $D^3 = D \backslash D^3_{out}$ is applied to train the mutant $\mathcal{F}^3$.

Specifically, for general DL tasks, we construct the model mutant based on the above strategy, and for categorical tasks, we discard data for each category to construct the mutants.

3.3 Fault Feature Extraction

After we generate mutants that amplify the differences between clean and faulty data, we extract a series of fault features for each data sample. The fault features are listed in table 1, which can be categorized into three classes: selection state (SS), output result (OR), and prediction loss (PR).

The selection state (SS) indicates whether corresponding data belong to a sparse feature in the training set, i.e., whether we apply it for model mutation. For each data (x, y) in the original training set, the selection state is marked as 1 if the model mutant applied it for training, otherwise, it is marked as 0. Therefore, the selection state of each model mutant is a discrete one-dimensional vector. We use the characteristic function $\mathbb{1}_X$ to indicate the selection state, and X is regarded as the training dataset. The output result (OR) represents the output vector of the DNN model when predicting each data, which reveals the execution differences between mutants and the original model. Thus, the OR features are continuous vectors with the same dimension as the DNN model's output. For prediction loss (PL) features, we adopt the same loss function used in the model optimization process. The loss function may be variant for different models and tasks, but we can determine the unique loss function $\mathcal{L}$ applied for a specific $\mathcal{F}$. The PL features are continuous one-dimensional variables. Compared to OR features which preserve detailed differences between mutants, PL features directly indicate the gap between prediction and label.

In summary, these fault features can be easily extracted through prediction, and reveal the behavior differences from diverse aspects. Because these features are general attributes of a DNN model, our approach could be applied to DNN datasets trained on arbitrary model structures or loss functions.

3.4 Dynamic Data Fault Localization

For a dataset under review, without a standard reference dataset, trusted clean data and real faults are both scarce. Meanwhile, due to the differences in data collection methods, data sources, annotator labeling habits, etc., data faults are distributed differently among various datasets. As a result, designing a universal suspiciousness score based solely on static scores (fault features) is not appropriate for detecting diverse data faults.

Therefore, inspired by traditional fault localization techniques [18, 50, 85], we choose to model a dynamically updated logistic regression model to predict the suspiciousness score. Denoting the employed model as *Susp*, the feature set of data sample $d = (x, y)$ extracted from table 1 as $F(d) = \{f_1(d), \ldots, f_m(d)\}$, and the weights correspond to each variable as $W = \{w_0, w_1, \ldots, w_m\}$, the fault probability computed by logistic function [20] is:

$$Susp(F(d)) = \frac{1}{1 + e^{-(w_0 + \Sigma_{i=1}^m w_i f_i(d))}}.$$

With a set of positive and negative samples, the internal weights W of the *Susp* could be optimized. The basic idea is to adjust the *Susp* using already checked data, which is a natural and convenient process for data collection. In the following, we will explain how to initialize and dynamically update the *Susp*.

Initialization. Initially, the entire dataset has not been checked by workers, and thus we do not have enough reliable positive (data fault) and negative (clean data) samples for training the *Susp*. To overcome this issue, we use the following strategies to initialize the *Susp*: 1) generate *artificial positive samples* by constructing noisy vectors; 2) prepare *artificial negative samples* by randomly sampling from the original dataset. The reason is two-fold: 1) noisy vector distributes more outlier than real data faults, making them effective for macroscopic modeling of fault features; 2) our method is based on the assumption that the majority of the dataset is correctly labeled, and thus most of the raw data are clean. Supposing the artificial positive and negative data are D^0_p and D^0_n respectively. Their corresponding fault features are denoted as $F(D^0_p), F(D^0_n)$. Therefore, without acquiring any manual feedback, we train the *Susp* model on these artificial samples to initialize it. Then the initialized *Susp* computes the suspiciousness score for each data sample $(x, y) \in D$ and ranks them in descending order.

Updating. Based on the trained *Susp*, all unchecked data are ranked, and a batch of high suspiciousness score data is provided to the annotators. With the assistance of data collection platforms [2, 9], workers can review and report incorrect or improper data samples. The platform allows testers to analyze the results and determine the next batch of data to be checked. Inspired by online learning [16], we propose a dynamic data ranking workflow that adjusts the *Susp* based on feedback from annotators. The model adapts to the specific distribution patterns of the current dataset, producing a more accurate ranking for subsequent data. For the k-th iteration, our goal is to sample a subset D^k for manual checking. First, we take the union of previously checked subsets $D^1, \ldots, D^{k-1}$, denoted as $D^{1,k-1}$. The identified data faults in $D^{1,k-1}$ are flagged as *real positive samples* ($D^{1,k-1}_p$), other samples are flagged as *real negative*

Table 2: Experiment combinations conducted in this paper.

#ID	Dataset	Model	Domain	Task	Fault Type
1	MNIST	LeNet-1	Image	Classify digits	Random Label Noise
2	MNIST	LeNet-1	Image	Classify digits	Specific Label Noise
3	MNIST	LeNet-1	Image	Classify digits	Random Data Noise
4	MNIST	LeNet-1	Image	Classify digits	Specific Data Noise
5	MNIST	LeNet-5	Image	Classify digits	Random Label Noise
6	MNIST	LeNet-5	Image	Classify digits	Specific Label Noise
7	MNIST	LeNet-5	Image	Classify digits	Random Data Noise
8	MNIST	LeNet-5	Image	Classify digits	Specific Data Noise
9	CIFAR10	ResNet-20	Image	Classify objects	Random Label Noise
10	CIFAR10	ResNet-20	Image	Classify objects	Specific Label Noise
11	CIFAR10	ResNet-20	Image	Classify objects	Random Data Noise
12	CIFAR10	ResNet-20	Image	Classify objects	Specific Data Noise
13	CIFAR10	VGG-16	Image	Classify objects	Random Label Noise
14	CIFAR10	VGG-16	Image	Classify objects	Specific Label Noise
15	CIFAR10	VGG-16	Image	Classify objects	Random Data Noise
16	CIFAR10	VGG-16	Image	Classify objects	Specific Data Noise
17	AGNews	LSTM	Text	Classify corpus	Random Label Noise
18	AGNews	LSTM	Text	Classify corpus	Specific Label Noise
19	AGNews	LSTM	Text	Classify corpus	Random Data Noise
20	AGNews	LSTM	Text	Classify corpus	Specific Data Noise
21	AGNews	BiLSTM	Text	Classify corpus	Random Label Noise
22	AGNews	BiLSTM	Text	Classify corpus	Specific Label Noise
23	AGNews	BiLSTM	Text	Classify corpus	Random Data Noise
24	AGNews	BiLSTM	Text	Classify corpus	Specific Data Noise
25	MTFL	TCDCNN	Image	Face Alignment	Label Noise
26	MTFL	TCDCNN	Image	Face Alignment	Data Noise

samples ($D_n^{1,k-1}$). The model *Susp* is then trained using $F(D_p^{1,k-1})$ and $F(D_n^{1,k-1})$. We apply the updated *Susp* to compute the suspiciousness score for all remaining unchecked data $(x, y) \in D \backslash D^{1,k-1}$ and rank them in descending order. The subset D^k is sampled by the updated *Susp*. Utilizing the feedback of previously checked data, this process is repeated until the review budget is reached, and the final data queue for manual checking is dynamically generated.

4 EXPERIMENT SETTINGS

This section presents the experimental setup, including the description of experiment subjects, baseline methods, parameter settings, and evaluation metrics.

4.1 Experiment Subjects

We employ 26 combinations of datasets, DNN models, and data fault types as experiment subjects. Table 2 lists all combinations that show the diversity in three aspects:

a) **Data types and tasks.** We apply DFauLo to two image classification datasets MNIST and CIFAR-10, one text dataset AgNews, and one face alignment dataset MTFL. MNIST [46] is a 10-class handwritten digits dataset, which contains 60,000 28×28 greyscale images for training and 10,000 for testing. CIFAR-10 [43] is a colored 10-class dataset, and each input is a 3-channel 32×32 image. AGNews [91] is a subset of AG's corpus of news articles constructed by assembling titles and description fields of articles from the 4 largest classes. Multi-Task Facial Landmark (MTFL) dataset [4] contains 12,995 face images with five coordinates of facial landmarks. The first three datasets (i.e., ID 1~24) are constructed for classification tasks, and the last dataset MTFL (ID 25&26) is for regression tasks.

b) **Network structures.** We implement 7 models with different network structures in the experiment, including 5 convolutional neural networks (CNNs) for image domain datasets (i.e., LeNet-1, LeNet-5 [45], ResNet-20 [32], VGG-16 [73], and TCDCNN [94]), as well as 2 classical recurrent neural networks (RNNs) for text domain datasets (i.e., LSTM [35] and BiLSTM [31]).

c) **Fault types.** Since data faults have different forms, for each model&dataset combination, we use multiple ways to simulate different types of data faults. As defined in Definition 1, we consider two types of data faults (i.e., *label noise* and *data noise*). Additionally, based on the observation that the distribution of data faults impacts the classification effectiveness [28], we also simulate two types of data distributions (i.e., symmetric and asymmetric). Symmetric noise, also known as random noise, supposes the faults are uniformly distributed among each class [28]. For *random label noise*, we randomly select 5% data from each class, and change their label into a random label other than the original one; for *random data noise*, we randomly select 5% data from each class, and replace their inputs with irrelevant data. In practice, label noise often exists between similar classes and data noise can be associated with only a specific class rather than all classes. To simulate this phenomenon, we introduce asymmetric noise. For *specific label noise*, we select 5% data from two classes and exchange their labels; for *specific data noise*, we randomly select 5% data from one class and replace it with irrelevant data. Let $A \rightarrow B$ represent that we use dataset A to replace data in dataset B. We apply 4 famous datasets in our experiments to inject data faults, denoted as CIFAR-10 $\leftrightarrow$ MNIST, IMDb [7] $\rightarrow$ AGNews, and COCO [54] $\rightarrow$ MTFL.

4.2 Baselines

We employ seven baselines, including three learning with noisy label (LNL) methods CleanLab [64], SemiFeat [96] and NCNV [47], two test prioritization techniques DeepGini [27] and DeepState [57], one advanced outlier detection method DIF [84], and one model uncertainty metric (denoted as Uncertainty [29, 60]). CleanLab is an open-sourced label issue detection tool based on the confident learning algorithm. It can be applied to classification datasets with any ML models and data types. SemiFeat is a data-centric noisy label clean technique aimed at identifying corrupt data labels in datasets without relying on training a model. NCNV is a method called Neighborhood Collective Noise Verification, which measures the degree of inconsistency to filter out the noisy data. DIF is a novel outlier detection method, which applies neural networks and isolation forests to detect outliers. Test prioritization is designed to select the data predicted incorrectly by the model. Since DNNs with good generalization should also make incorrect predictions on faulty data, these methods are valuable as baselines for evaluating their effectiveness in detecting data faults. DeepGini is a simple but effective test prioritization method for DNN with the Softmax output layer. DeepState is a test selection method specifically designed for recurrent neural networks (RNNs), which selects error tests based on the internal states of the RNNs. The above methods cannot be applied to the regression model, and we introduce a general method Uncertainty as a baseline method, which evaluates the model uncertainty for DNN with a dropout layer. Other prioritization and LNL techniques are omitted because they require to be trained on a trusted training set [51, 80, 92] or do not involve a rank for data inspection [41, 49, 89]. Besides, other detection techniques such as concept drift detection [58] which do not focus on wrongly labelled data are not considered either.

4.3 Parameter Settings

To implement DFauLo, we need to set some hyper-parameters. First, for model mutation strategy, the implementation of VAE is based on the open-sourced Python tool PyOd [95]. The `remove_ratio` α is set as 5%. We mutate the original model by directly retraining the original model, the `retraining_epoch` is set as 10, and the

`learning_rate` is the same as the setting of model training. To simulate the updating process of *Susp*, we set the `iteration_batchsize` as 200 and update the model *Susp* per batch to prioritize the next batch of unchecked data. The maximum ratio used for updating *Susp* (i.e., review budget) is 20%; after that, we rank all the unchecked data based on the last *Susp*. Finally, the whole data rank is reordered based on the iteration batch to evaluate the effectiveness of DFauLo.

4.4 Evaluation Metrics

We employ three metrics, i.e., **PoBL(10%)**, **RAUC**, and **AUC-ROC**, to evaluate the effectiveness of baselines and DFauLo. PoBL(10%) refers to the proportion of located data faults to all data faults in the top 10% data, it measures the effectiveness of localization with a fixed manual budget. Furthermore, to measure the overall localization effectiveness, we plot the localization performance of each subject into a figure, where the x-axis is the ratio of data ranked by DFauLo or baselines, and the y-axis is the ratio of fault already detected. We also plot the theoretically best localization performance in which all faults are prioritized in front of other data. We calculate the ratio of the area under the curve for the data fault localization approach to the area under the curve for the ideal data rank as another evaluation metric, named RAUC [14, 80]. RAUC ranges from 0 to 1, and a larger RAUC value indicates better performance. Considering the fact that the fault localization problem can be naturally modeled as a binary classification task, we also employ the AUC-ROC metric to evaluate the performance. Different from RAUC, AUC-ROC indicates the effectiveness of selecting a threshold for classifying positive and negative samples.

5 RESULTS AND DISCUSSIONS

In the experiment, we mainly focus on the following four research questions (RQs):

- **RQ1: Effectiveness.** How effective is DFauLo in locating data faults in the DL datasets?
- **RQ2: Ablation Study.** What is the impact of each component in the DFauLo design on the effectiveness?
- **RQ3: Efficiency.** What is the computational efficiency of DFauLo compared to baselines?
- **RQ4: Potentials.** To what extent can DFauLo improve the performance of DNN models via locating the data faults?

Additionally, to evaluate the effectiveness in real application scenarios, we conduct a case study using benchmark datasets to explore the DFauLo's superiority in locating natural existence data faults.

5.1 RQ1: Effectiveness

Tables 3 and 4 show the results on subjects with classification and regression tasks, respectively. To mitigate bias, we run the whole experiment five times and report the average results. In table 3 and 4, we list the results based on data fault types. Additionally, figure 3 shows localization curves indicating the proportion of located faults versus the proportion of prioritized data.

Results. LNL techniques, including CleanLab, NCNV, and SemiFeat, perform well (RAUC>0.9) in most subjects with label noises (table 3 row 1-12), which is significantly better than other baselines. For datasets with random data noises (row 13-18), most baselines show competitive performance (RAUC≥0.812), but NCNV and DIF performs poor on LeNet-5 (RAUC=0.514) and AGNews (RAUC=0.522) respectively. Compared with CNN models, the results of all methods on RNN models are less effective. The RNN-specific method DeepState performs better on data noise than label noise. In table 4, Uncertainty shows poor performance in all evaluation metrics, PoBL(10%) (≤0.1), RAUC (≤0.517) and AUC-ROC (≤0.504). For most of the experiment subjects, the RAUC and AUC-ROC metrics of DFauLo are more effective and stable (all results exceed 0.91) compared to baselines. Additionally, for the evaluation metric PoBL(10%), DFauLo even reaches the maximum value of 1.00 on some subjects, which means it locates all data faults in the top 10% list. It can also be observed that the curves of DFauLo in figure 3 are closer to the theoretically best curves than other baselines.

Discussion. According to the results, DFauLo outperforms other baselines in both classification and regression datasets, with consistent effective results. The results suggest that Uncertainty has no significant effect on locating data faults, which is likely due to the model's overfitting to noisy data in the training set. For the first three groups of fault types (table 3 row 1-18), LNL techniques are more effective than other baselines, and CleanLab performs better than other LNL techniques. All LNL techniques failed to demonstrate effectiveness and perform even worse than random ranking for specific data noises. On the other hand, we notice that outlier detection method DIF is good at locating image data noises, but shows bad performance on locating label noises and text data noises. Two test prioritization methods (DeepGini and DeepState) are effective in identifying random data noise, but are not sufficiently sensitive to label noise. Considering the fact that all of test prioritization techniques are designed for selecting the unlabelled data for manual labelling, we speculate this experiment result fits their design choice well. Besides, NCNV requires to train DNN models with specific designed process, the default training setting may not convergence for NCNV, this property illustrate its poor results on MNIST&LeNet-5. Our findings indicate that existing methods based on static data ranking become useless in finding faults in dataset with more diverse data faults. Additionally, combined with the results of table 3 and table 4, DFauLo shows stable effectiveness in all fault types, dataset domain, model structure and task types, demonstrating its superior generalizability.

5.2 RQ2: Ablation Study

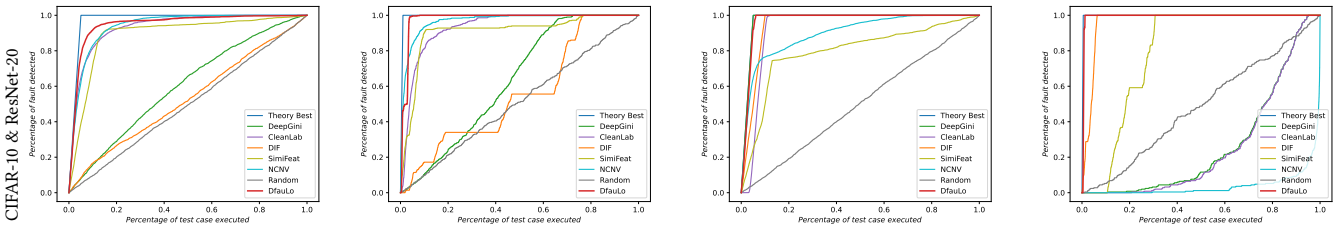
We construct an ablation study to investigate the effectiveness of DFauLo's designs. Specifically, we evaluate the localization performance of the suspiciousness function *Susp* without utilizing all model mutants and without dynamic iteration respectively. To mitigate bias and reach a statistically significant result, we repeat experiments on each subject 30 times. Also, we employ the Wilcoxon rank-sum test [34] and Cliff's δ analysis [70] to analyze whether there are significant differences between the ablation setting and the design of DFauLo. Following the conventions of previous work [63, 72], if the rank-sum significance level $p < 0.05$ and effect size $|\delta| \geq 0.474$, we mark the corresponding setting is significantly worse (or better) than DFauLo. Limited by space, we select 6 experiment subjects and report the average RAUC in table 5.

Results. For most subjects, the initialized *Susp* model (without feedback) shows competitive performance (RAUC $\geq$ 0.8958) compared to the best baselines in table 3; and for challenging experimental settings, such as #ID 24 (specific data noise in AGNews&BiLSTM), without the aid of feedback, the initialized *Susp*

Table 3: The overall effectiveness of DFauLo and baselines on classification datasets with different fault types.

Fault Type	Dataset	Model	PoBL(10%) DFauLo	PoBL(10%) Baselines						RAUC DFauLo	RAUC Baselines						AUC-ROC DFauLo	AUC-ROC Baselines					
				CleanLab	SemiFeat	NVNC	DIF	DeepGini	DeepState		CleanLab	SemiFeat	NVNC	DIF	DeepGini	DeepState		CleanLab	SemiFeat	NVNC	DIF	DeepGini	DeepState
Random Label Noise	MNIST	LeNet1	**0.998**	0.981	0.991	0.995	0.589	0.112	-	**0.997**	0.993	0.983	0.997	0.771	0.542	-	0.998	**0.998**	0.982	0.997	0.758	0.518	-
	MNIST	LeNet5	**0.991**	0.975	0.942	0.154	0.589	0.102	-	0.991	**0.993**	0.968	0.587	0.771	0.519	-	0.992	**0.993**	0.966	0.576	0.758	0.506	-
	CIFAR10	ResNet20	**0.917**	0.815	0.719	0.830	0.166	0.157	-	**0.973**	0.957	0.915	0.964	0.543	0.616	-	**0.972**	0.956	0.914	0.963	0.535	0.606	-
	CIFAR10	VGG16	**0.945**	0.817	0.599	0.881	0.166	0.130	-	**0.977**	0.955	0.909	0.970	0.543	0.546	-	**0.976**	0.954	0.911	0.970	0.535	0.534	-
	AGNews	LSTM	**0.875**	0.863	0.782	0.864	0.000	0.454	0.297	0.963	**0.968**	0.890	0.965	0.464	0.797	0.633	0.962	**0.968**	0.867	0.964	0.447	0.792	0.636
	AGNews	BiLSTM	**0.875**	0.859	0.759	0.894	0.000	0.472	0.282	0.951	0.967	0.877	**0.972**	0.464	0.815	0.674	0.947	0.966	0.857	**0.971**	0.447	0.810	0.666
Sepcific Label Noise	MNIST	LeNet1	**1.000**	0.995	0.949	**1.000**	0.517	0.142	-	**0.995**	0.983	0.955	0.997	0.728	0.701	-	**0.996**	0.990	0.950	0.998	0.710	0.686	
	MNIST	LeNet5	**1.000**	0.989	0.958	0.009	0.517	0.144	-	0.974	**0.976**	0.948	0.617	0.728	0.663	-	**0.996**	0.976	0.944	0.614	0.710	0.661	
	CIFAR10	ResNet20	**1.000**	0.810	0.900	0.934	0.172	0.102	-	**0.985**	0.937	0.911	0.973	0.561	0.636	-	**0.987**	0.937	0.916	0.974	0.536	0.634	
	CIFAR10	VGG16	**1.000**	0.952	0.826	0.948	0.172	0.152	-	**0.994**	0.948	0.916	0.975	0.561	0.620	-	**0.998**	0.948	0.924	0.976	0.536	0.618	
	AGNews	LSTM	0.806	0.676	0.511	**0.853**	0.000	0.480	0.318	**0.955**	0.886	0.638	0.950	0.248	0.812	0.692	**0.951**	0.884	0.731	0.949	0.449	0.809	0.693
	AGNews	BiLSTM	**0.925**	0.835	0.761	0.844	0.000	0.301	0.243	**0.980**	0.950	0.802	0.939	0.248	0.750	0.695	**0.980**	0.949	0.851	0.937	0.449	0.746	0.686
Random Data Noise	MNIST	LeNet1	**0.999**	0.970	0.744	0.950	0.912	0.979	-	**0.996**	0.972	0.858	0.984	0.932	0.996	-	**0.996**	0.986	0.847	0.984	0.927	0.992	-
	MNIST	LeNet5	**0.999**	0.876	0.701	0.094	0.912	0.781	-	**1.000**	0.969	0.847	0.514	0.932	0.950	-	**0.999**	0.968	0.845	0.501	0.927	0.949	-
	CIFAR10	ResNet20	**1.000**	0.910	0.499	0.765	0.999	**1.000**	-	0.997	0.954	0.817	0.923	0.974	**1.000**	-	0.998	0.952	0.816	0.921	0.973	**1.000**	-
	CIFAR10	VGG16	**0.999**	0.683	0.420	0.730	**0.999**	**0.999**	-	0.999	0.935	0.812	0.935	0.974	**1.000**	-	0.999	0.933	0.801	0.934	0.973	**1.000**	-
	AGNews	LSTM	**0.944**	0.925	0.667	0.571	0.121	0.941	0.726	0.980	0.968	0.832	0.923	0.522	**0.987**	0.873	0.981	0.967	0.807	0.921	0.511	**0.987**	0.869
	AGNews	BiLSTM	0.966	0.956	0.669	0.494	0.121	**0.972**	0.567	0.986	0.971	0.838	0.917	0.522	**0.993**	0.897	0.990	0.971	0.808	0.914	0.511	**0.993**	0.895
Specific Data Noise	MNIST	LeNet1	**0.997**	0.071	0.212	0.990	0.915	0.062	-	**0.998**	0.255	0.259	0.989	0.884	0.251	-	**0.997**	0.671	0.563	0.987	0.910	0.655	-
	MNIST	LeNet5	**0.977**	0.143	0.134	0.000	0.915	0.108	-	**0.993**	0.423	0.194	0.075	0.884	0.408	-	**0.993**	0.421	0.521	0.072	0.910	0.405	-
	CIFAR10	ResNet20	**1.000**	0.000	0.000	0.000	**1.000**	0.000	-	**0.999**	0.264	0.799	0.041	0.969	0.271	-	**0.996**	0.261	0.447	0.037	0.952	0.268	-
	CIFAR10	VGG16	**1.000**	0.440	0.352	0.000	**1.000**	0.344	-	**1.000**	0.852	0.859	0.132	0.969	0.814	-	**0.996**	0.850	0.682	0.129	0.952	0.812	-
	AGNews	LSTM	**0.645**	0.046	0.077	0.063	0.128	0.061	0.580	**0.920**	0.502	0.750	0.751	0.728	0.509	0.867	**0.919**	0.498	0.476	0.749	0.514	0.505	0.866
	AGNews	BiLSTM	0.654	0.119	0.057	0.095	0.128	0.141	**0.758**	**0.919**	0.567	0.694	0.643	0.728	0.574	0.913	0.912	0.564	0.478	0.640	0.514	0.571	**0.913**

* For each experiment subject, we **bold** the maximum evaluation results to highlight the technique with the best performance. DeepState can only be applied to RNN models.



(a) Random Label Noise (b) Specific Label Noise (c) Random Data Noise (d) Specific Data Noise

Figure 3: Data fault localization curves. X-axis indicates the percentage of prioritized data with the highest suspiciousness, and Y-axis indicates the percentage of located data faults.

Table 4: Data fault localization effectiveness on face alignment task of different fault types.

#ID	Fault Type	PoBL(10%) DFauLo	PoBL(10%) Uncertainty	RAUC DFauLo	RAUC Uncertainty	ROC-AUC DFauLo	ROC-AUC Uncertainty
25	Label Noise	**0.958**	0.089	**0.962**	0.481	**0.963**	0.467
26	Data Noise	**0.962**	0.096	**0.966**	0.517	**0.957**	0.504

Table 5: Average RAUC results of ablation study on 6 experiment subjects.

Models Applied	+ Feed back?	# ID 1	6	11	16	17	24
Input	×	0.9289	0.9372	0.9899	0.9958	0.8292	0.4661
	✓	0.9953	0.9593	**0.9979**	0.9997	0.9602	**0.9210**
Hidden	×	0.8931	0.9691	0.9207	0.7859	0.9146	0.4317
	✓	0.9953	0.9481	0.9686	0.9922	0.9576	0.8894
Output	×	0.9885	0.9544	0.9153	0.8965	0.8757	0.1904
	✓	0.9955	0.9254	0.9729	0.9923	0.9568	0.8948
ALL	×	0.9547	0.9613	0.9712	0.9679	0.8958	0.4276
	1%	0.9675	0.9056	0.9901	0.9997	0.8915	0.9150
DFauLo	✓	**0.9973**	**0.9707**	**0.9966**	**0.9998**	**0.9625**	**0.9176**

* If the result wins DFauLo, we **bold** the corresponding result, and if it loses to DFauLo, we mark it as gray.

also shows a poor result (RAUC = 0.4276) like other static baseline methods. Besides, employing all three model mutants to construct *Susp* model can more effectively locate data faults, and DFauLo wins 16 out of 18 comparisons (rows 2,4,6 compared to the last row). Even with only 1% human feedback, the localization performance can be improved, especially for difficult data fault types (the average RAUC of #ID 24 is improved from 0.4276 to 0.9150).

Discussion. First, using DNN model mutants generated from different dimensions of the model provides DFauLo with effective fault features for locating diverse types of faults. Besides, when *Susp* models leverage no human feedback, the result can be seen as a static technique. In this case, such "Static-FauLo" exhibits both performance and bottlenecks similar to state-of-the-art static baseline methods. The dynamic iterative design of DFauLo is the key to breaking through the predicament of static techniques. For those data fault types that are difficult to analyze and identify by static techniques, DFauLo can promptly utilize feedback to learn unknown features of data faults. Finally, taking advantage of more feedback can better improve DFauLo's performance. Thus introducing the dynamic iterative design into online crowdsourcing tasks can maximize the DFauLo's advantages over other static methods.

5.3 RQ3: Efficiency

We collect and report the average execution time of different methods in table 6. On the one hand, we compare DFauLo with other baseline methods; on the other hand, since DFauLo composes multiple steps, we also collect the separate computation time of DFauLo. The top half of the table reports the time cost by baselines, and the last half of the table reports the overall (Row ALL) time and separate time cost for each operation of DFauLo.

Table 6: Average execution time (seconds) of baseline methods and DFauLo.

	Dataset	MNIST		CIFAR-10		AgNews		MTFL
	Model	LeNet-1	LeNet-5	ResNet-20	VGG-16	LSTM	BiLSTM	TCDCNN
Baseline	DeepGini	26	38	182	231	217	332	/
	DeepState	/	/	/	/	280	270	/
	Uncertainty	/	/	/	/	/	/	72
	DIF	256	259	261	263	470	486	/
	CleanLab	1983	2613	5752	4663	2301	5029	/
	SemiFeat	1664	1627	1539	1456	3460	3481	/
	NCNV	4000	3180	7398	6923	4424	5264	/
DFauLo	**ALL**	990	1000	2926	2183	1765	2306	284
	Select Subset	180	170	511	324	296	335	102
	Mutation&Extraction	805	826	2411	1855	1456	1958	180
	Initialize *Susp*	0.4	0.3	0.6	0.9	0.8	0.9	1.8
	Update *Susp*	4.0	3.7	3.5	3.0	11.0	11.0	0.5

Results. Table 6 shows DeepGini, DeepState, Uncertainty, and DIF take less than 500 seconds for all experiments, which are the most efficient techniques. Notwithstanding being effective in RQ1, LNL techniques (CleanLab, NCNV, and SemiFeat) manifest poor efficiency. While DFauLo's efficiency may not be the best, it is acceptable, as all subjects can be ranked within an hour (< 3000s). Model mutation and feature extraction exhaust the majority of time in DFauLo, while initializing and updating the *Susp* model is rather efficient, which only requires less than 12 seconds.

Discussion. DeepGini and DeepState are efficient due to their use of simple model behaviors during the prediction and calculation of static scores for ranking. However, the original model may overfit the training dataset and such static scores are not effective for data fault localization. CleanLab adopts cross-validation to train multiple models and compute confusion matrices, which requires much larger training epochs and has limitations in terms of applicable dataset types. NCNV needs to train two models with a specific method from scratch, which takes the longest training time. SemiFeat does not need to train the model, but it expends much more computation on data processing and repeated execution. DFauLo requires retraining the original model, but only 10 epochs are sufficient, making its efficiency acceptable. In terms of dynamic iteration efficiency, updating *Susp* after each iteration is quite efficient (0.5-11s), as all fault features are extracted statically beforehand. Therefore, incorporating DFauLo into the data correction workflow would not consume excessive time for testers.

5.4 RQ4: Potentials

Intuitively, if we can locate faults in the dataset and correct them, we can obtain a clean dataset. Therefore, we are curious about whether we can remedy some defective behaviors by retraining the model with a cleaned dataset. Based on the data ranking computed by each method, we correct the data faults in the top 5% as follows: inputs with incorrect labels are corrected, and irrelevant inputs are deleted. Then we retrain the original model and compare the accuracy fluctuations between different methods. In addition, we also analyze the effect of combining DFauLo with other training methods in table 8. To evaluate whether DFauLo can enhance the effectiveness of such methods, we implement two state-of-the-art model-centric LNL methods, namely NLNL [39] and DivideMix [48], and compare the model accuracy with and without DFauLo correction. We prepared two training sets for each fault type, one is with data faults injected, and the other is corrected by DFauLo for the top 20% data.

Table 7: Repairing defective DNNs with each method.

		Fault Type	Ori.	DFauLo	Baselines					
					CleanLab	SemiFeat	NVNC	DIF	DeepGini	DeepState
MNIST	LeNet1 Label	Rand.	95.63	**96.91**	96.87	96.87	96.76	96.07	96.07	-
		Spec.	94.92	**96.02**	95.49	95.56	95.60	95.31	95.04	-
	LeNet1 Data	Rand.	95.20	**96.90**	96.34	96.02	96.42	96.56	96.81	-
		Spec.	94.91	**96.59**	95.35	95.83	95.66	96.52	96.10	-
	LeNet5 Label	Rand.	90.94	**94.66**	94.18	94.58	92.44	94.40	94.55	-
		Spec.	91.65	**94.69**	93.30	93.63	93.13	94.48	94.47	-
	LeNet5 Data	Rand.	90.15	**94.81**	93.13	93.31	93.21	94.35	94.52	-
		Spec.	90.16	**94.75**	92.61	93.10	94.00	94.40	94.21	-
CIFAR-10	ResNet20 Label	Rand.	73.61	**80.93**	79.96	77.44	79.88	74.90	74.76	-
		Spec.	76.90	**82.93**	82.08	82.31	82.21	81.93	81.12	-
	ResNet20 Data	Rand.	77.02	82.18	82.37	82.41	**82.43**	81.71	81.47	-
		Spec.	74.96	**83.06**	82.08	82.73	82.36	82.95	82.86	-
	VGG16 Label	Rand.	73.52	**86.15**	84.62	83.31	85.90	83.35	82.27	-
		Spec.	77.38	85.90	**86.44**	85.52	86.18	83.99	85.60	-
	VGG16 Data	Rand.	78.43	**86.37**	85.86	85.70	86.31	86.36	85.69	-
		Spec.	73.33	**86.48**	85.94	85.99	86.32	86.36	86.15	-
AGNews	LSTM Label	Rand.	88.96	89.02	**89.50**	88.53	89.32	86.34	87.06	87.06
		Spec.	89.14	**89.22**	89.06	89.19	89.13	87.88	88.51	88.51
	LSTM Data	Rand.	90.39	90.42	**91.22**	90.88	90.92	90.09	89.69	89.69
		Spec.	83.97	**84.65**	83.76	82.09	83.03	80.63	83.05	83.05
	BiLSTM Label	Rand.	90.07	**90.17**	90.05	89.53	89.89	87.47	87.92	87.92
		Spec.	89.84	**90.72**	90.42	90.53	90.42	88.59	88.77	88.77
	BiLSTM Data	Rand.	89.46	89.94	**90.50**	90.25	90.06	89.93	90.17	90.17
		Spec.	81.75	**84.32**	83.06	82.43	84.23	82.48	81.85	81.85
MTFL	Label Noise		0.1058	**0.1054**	**Baseline**				0.1059	
	Data Noise		0.1062	**0.1056**	Uncertainty				0.1067	

* For each subject, we **bold** the top-1 boxes with the highest accuracy increment.

Table 8: Test accuracy on CIFAR-10&ResNet-20 of different training methods with and without DFauLo.

Training Method		Label Noise		Data Noise	
		Random	Specific	Random	Specific
Ori.	\	73.61%	76.90%	77.02%	74.97%
	+DFauLo	78.64%	80.64%	78.24%	78.25%
NLNL	\	81.95%	82.23%	82.32%	82.80%
	+DFauLo	83.13%	82.46%	82.14%	82.65%
DivideMix	\	81.54%	81.93%	81.17%	82.06%
	+DFauLo	82.23%	82.22%	82.04%	82.13%

Results. Table 7 reports the model accuracy fluctuations when retraining the original DNN model with different fault localization methods. For all combinations, retraining the model with the DFauLo corrected dataset could improve the accuracy of the model. Consistent with the results of RQ1, compared with other baseline methods, DFauLo achieve the highest accuracy improvement in most subjects (21 out of 26 subjects). Table 8 presents the effectiveness of DFauLo combined with LNL methods. For the DivideMix method trained on the noisy dataset, the model accuracy is 81.54%, 81.93%, 81.17%, and 82.06%, respectively. If DivideMix is applied on a dataset cleaned by DFauLo, the model accuracy increases to 82.23%, 82.22%, 82.04%, and 82.13%, respectively.

Discussion. Based on table 7, we observe that most defective DNN models can be well repaired if the data faults are corrected. Referring to table 3, we draw the conclusion that DFauLo has better potential in repairing defective models via locating more data faults. Additionally, the accuracy of different types of data fault indicates that specific data noise is most harmful to the performance. Based on the results of table 8, DFauLo can assist in the LNL methods to overcome the defects in the dataset and improve the model performance. This phenomenon also demonstrates the necessity of debugging the dataset before model training.

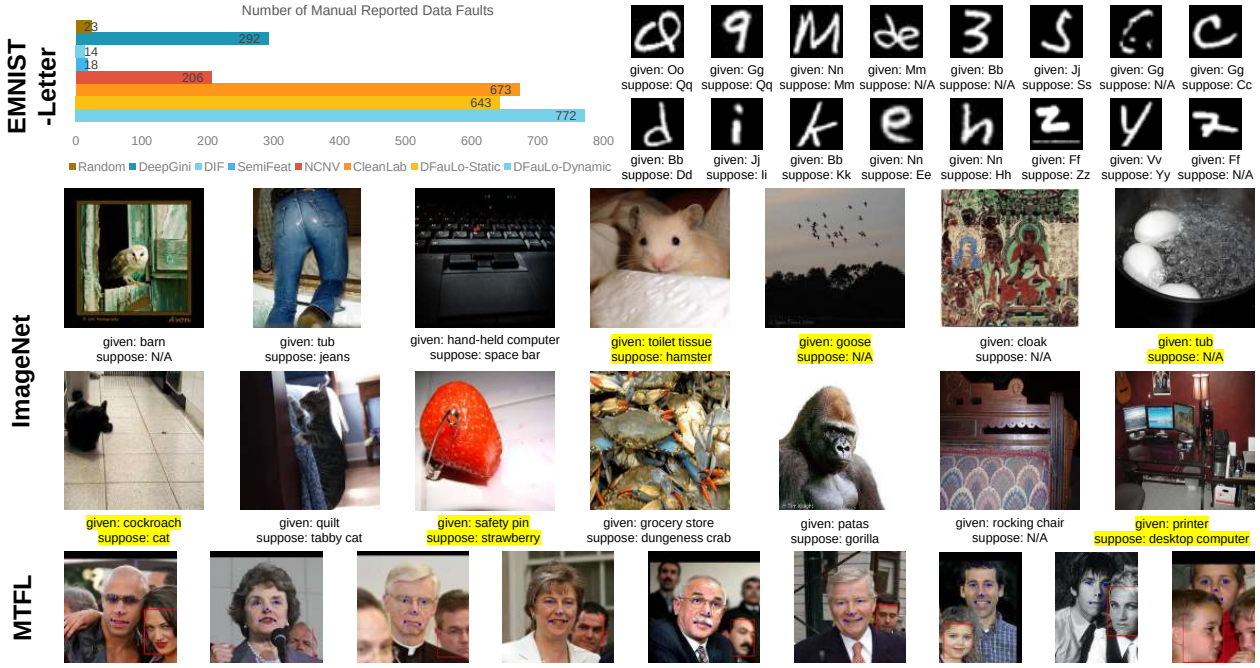


Figure 4: Data faults detected by DFauLo in widely applied benchmark datasets.

5.5 Case Study on Benchmark Datasets

5.5.1 Comparing DFauLo with Baselines. To prove the superiority of DFauLo over other baseline methods, we deploy all methods on the same dataset, EMNIST-Letter [22]. EMNIST-Letter is a dataset containing 12,4800 samples of 26 handwritten letters. The SOTA DNN model of EMNIST-Letter, WaveMix [5], is employed for methods that require a DNN model. For each baseline, the top 1% of the data is prioritized for manual inspection. For our method, we collect 1% data prioritized by the initialized *Susp* model (noted as Static-FauLo), as well as 1% data dynamically sampled by DFauLo. Each data sample is allocated for 5 independent crowd workers to check. Each sample is scored on a scale of $-2, -1, 0, 1, 2$, with higher scores indicating higher quality data, and workers are asked to diagnose the type of fault and provide alternative labels. If a sample receives three or more negative scores, we mark it as a fault and use them for updating the *Susp*. The top left of figure 4 reports the number of data faults found by each method. Detailed subsets of each method with the scores and diagnostic information for each data sample can be found on our website.

Compared with the estimated data fault rate (1.84%), only two baseline techniques (DIF and NCNV) failed to prioritize real data faults. We assume this may be caused by their limited design intentions (only designed to identify outlier noises or data with incorrect labels). In addition, CleanLab performs the best among all baseline methods, with a fault rate of 53.93% of the selected set. Consistent with the conclusions in RQ2, the initialized *Susp* (denoted as Static-FauLo) almost achieves the best static method performance (51.52% fault rate). And when we introduce the dynamic design, the DFauLo shows outstanding results than other methods. DFauLo locates 772 data faults out of 1248 manually inspected data, and the corresponding fault rate is 61.86%, which demonstrates the superiority of DFauLo in real-world application scenarios.

5.5.2 Applying DFauLo on Complicated Benchmark Datasets. After demonstrating the superiority of DFauLo in identifying EMNIST data faults, we further apply it to more complex benchmark datasets to uncover more diverse data faults. For the classification task, we conduct the case study on the large-scale ImageNet [24] dataset, utilizing a pre-trained model with ResNet-50 architecture provided by PyTorch [11]. we apply DFauLo on the validation set of ImageNet, e.g. ILSVRC2012 [6], which includes 1000 classes and 50 images for each class. Consistent with the procedure described above, after initializing the *Susp* model, the top 1% data samples are prioritized for 5 independent crowd workers to check. For the regression task, the MTFL [4] dataset is selected for the case study, we first follow the official suggestion to pre-process the images to get a face bounding box [13], and then apply DFauLo to locate data faults. Limited with resources, we repeat the dynamic iteration 3 times for ImageNet, and once for MTFL. Figure 4 presents some detected real faults in these benchmark datasets.

The supposed label may be a substitute for the given label or coexist with the given label. N/A represents that the corresponding image cannot be categorized into a specific class. In the MTFL dataset, we present the images in which the automatically generated bounding boxes mismatch the face alignment labels. Based on the results in figure 4, we find that both two fault types defined in Definition 1 actually exist in reality. Some input images lack recognizable features or have more than one object to classify. Other data have recognizable features but are labeled to the incorrect categories. For example, in the first row of figure 4, the first image is labeled as "barn", while the main object in the image is an owl. Though the background of the image may indeed be a barn, this input still lacks sufficient features of the barn for workers to label. However, "owl" is not a candidate label in ImageNet (ILSVRC2012). Thus the crowd workers assume it is irrelevant data which not belong to the dataset. Besides, if a suitable label exists, we recommend

workers suppose an alternative label. The second image in the first row, labeled as "tub", is reported to be better labeled as "jeans". The third image in the first row is labeled as "hand-held computer", while it can also be categorized into other classes, such as "space bar". Moreover, comparing the data faults between ImageNet and EMNIST, we observe that the complexity of the input and tasks also leads to more diverse data faults. Compared to the [65] study which used CleanLab to manually review half of ImageNet's data, DFauLo discovered many unnoticed data faults. Out of 1500 ImageNet data, 436 received negative scores from at least one worker in our study but were not prioritized by CleanLab as the top half of the possible faulty data. These newly found faults are highlighted in figure 4 and more detailed information is available on our website.

6 THREATS TO VALIDITY

We noticed some threats in our study. The internal threat of DFauLo comes from the errors committed by crowd-workers. The DFauLo leverages the outcomes of human feedback to dynamically update the model *Susp*, and imperfections in human feedback could negatively influence the effectiveness of the DFauLo. We mitigate this threat by evaluating DFauLo on benchmark datasets, which utilize authentic human feedback that inherently reduces inaccuracies and inconsistency for *Susp* updating. The external threat to validity lies in the subjects used in our experiment. The selection of datasets and simulated data faults could influence the performance of the results. To reduce this threat, we construct 26 model and dataset combinations with different data types and tasks, network structures, and fault types in the experiment. Moreover, we conduct a case study on benchmark datasets to investigate the performance of DFauLo in detecting real data faults. All of these results show consistent effective results of DFauLo. Another threat comes from the randomness of all localization techniques. First, the randomness of DNN prediction could influence the ranking of faulty data, which poses a threat to effectiveness comparison. Besides, the randomness when initializing the *Susp* model also poses a threat to the ablation study of DFauLo. To reduce this threat, we repeat all methods multiple times to collect average results and introduce statistical methods for analysis.

7 RELATED WORKS

This section reviews some related works in fault localization and mitigation for deep learning-based software testing.

Repairing DL software codes. Similar to traditional software testing, defects may also exist in the program codes of DL-based software. First, faults may exist in DL libraries and then be inherited by a concrete implementation of the DL model. Pham et al. [69] proposed *CRADLE* to detect and localize bugs in DL libraries with differential testing. Wang et al. [79] proposed *LEMON* to debug DL libraries by generating effective DL models via guided mutation. Wei et al. [82] proposed a fuzzing technique *FreeFuzz* to debug DL libraries based on open source APIs automatically. Second, code faults also exist in the model construct and training codes [36]. Wardat et al. [81] proposed *DeepLocalize* to identify the root cause (code) for DNN errors, and the internal values are analyzed in both feedforward and backpropagation phases to identify and locate the incorrect codes and parameters. Zhang et al. [90] introduced *AUTOTRAINER* to test and repair the mode training process.

Adjusting neuron weights. Neuron networks rely on numerous hidden neurons to make a prediction. Once the model finishes training and is deployed to the application scenario, its internal weights are frozen. Hence, compared to code fault, for a deployed (trained) DL model, more potential defects may hide in the trainable parameters of the neurons. Eniser et al. [25] proposed *DeepFault* to identify suspicious neurons whose weights are not calibrated correctly and impact the DNN performance. Sun et al. [76] proposed *CARE* to locate and modify the weights of suspicious neurons whilst maintaining general accuracy. To reduce defect inheritance in transfer learning, Zhang et al. [93] designed a relevant model slicing method *ReMoS* to avoid the model inheriting the neuron faults hidden in the teacher model. In addition, extensive deep learning testing papers attempt to improve DL model quality by retraining or fine-tuning the model, and the research subjects range from test selection [27] to test generation [56, 77].

Debugging training data. Most of the abovementioned techniques assume that we have a trusted dataset for testing and repairing the DL model. However, such an assumption is fragile and hard to guarantee. Li et al. [52] proposed *LTDD* to debug feature values in training data and improve the fairness of machine learning models. Northcutt et al. [64] proposed an open-source data debugging tool *CleanLab* to estimate and find label errors for classification datasets. Based on *CleanLab*, Northcutt et al. [65] conducted a large-scale empirical study on the test sets of 10 classical ML datasets and estimated there are at least 3.3% errors across the 10 datasets.

The relationship of DFauLo to the related work can be summarized into three folds: 1) it improves DL software quality in parallel with techniques that fix the code faults in the DL program; 2) it provides reliability guarantees for techniques that adjust the internal model weights with datasets; 3) instead of focusing on biased features which are hard to capture and repair, DFauLo is designed to locate buggy data that can be interpreted and corrected manually.

8 CONCLUSION

This paper proposes a dynamic data fault localization technique named DFauLo. Data faults in the deep learning dataset can be dynamically prioritized by DFauLo, and then workers can check and correct them without reviewing the whole dataset. We demonstrate the effectiveness of DFauLo on a wide range of experiment subjects. Moreover, we also apply DFauLo to widely-used benchmark datasets and detect various real data faults. Our technique complements the gap in data quality assurance for the testing of DL-based software systems, and we will explore automatically repairing these located data faults in the future.

9 DATA AVAILABILITY

All datasets used in the experiment can be easily accessed on their official websites. The DFauLo and experiment source codes are released at: https://zenodo.org/record/8266660. We have also released the detected data faults in the case study and remaining experiment results at: https://sites.google.com/view/dfaulo.

ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their constructive comments. This project was partially funded by the National Key R&D Program of China (2021ZD0112903) and the National Natural Science Foundation of China under Grant No. 61932012.

REFERENCES

[1] [n. d.]. Amazon Mechanical Turk. https://www.mturk.com/. (Accessed on 02/01/2023).

[2] [n. d.]. Amazon Mechanical Turk. https://www.mturk.com/product-details. (Accessed on 01/18/2023).

[3] [n. d.]. Confidence to Deploy AI with World-Class Training Data. https://appen.com/. (Accessed on 02/01/2023).

[4] [n. d.]. Facial Landmark Detection by Deep Multi-task Learning. http://mmlab.ie.cuhk.edu.hk/projects/TCDCN.html.

[5] [n. d.]. GitHub - pranavphoenix/WaveMix: 2D discrete Wavelet Transform for Image Classification and Segmentation. https://github.com/pranavphoenix/WaveMix. (Accessed on 06/07/2023).

[6] [n. d.]. ImageNet. https://image-net.org/challenges/LSVRC/2012/index. (Accessed on 09/02/2022).

[7] [n. d.]. IMDb Datasets. https://www.imdb.com/interfaces/.

[8] [n. d.]. MIT study finds 'systematic' labeling errors in popular AI benchmark datasets. https://venturebeat.com/business/mit-study-finds-systematic-labeling-errors-in-popular-ai-benchmark-datasets/.

[9] [n. d.]. Prolific · Developers. https://www.prolific.co/developers. (Accessed on 01/29/2023).

[10] [n. d.]. Prolific · Quickly find research participants you can trust. https://www.prolific.co/. (Accessed on 02/01/2023).

[11] [n. d.]. ResNet50 | PyTorch. https://pytorch.org/hub/nvidia_deeplearningexamples_resnet50/. (Accessed on 09/02/2022).

[12] [n. d.]. wengshihao/DFauLo: Dfaulo V1.3 | Zenodo. https://zenodo.org/record/8266660. (Accessed on 08/20/2023).

[13] [n. d.]. zhzhanp/TCDCN-face-alignment: Matlab implementation of facial landmark detection by deep multi-task learning. https://github.com/zhzhanp/TCDCN-face-alignment. (Accessed on 09/02/2022).

[14] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2009. Spectrum-based multiple fault localization. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 88–99.

[15] Brandon Amos, Bartosz Ludwiczuk, Mahadev Satyanarayanan, et al. 2016. Openface: A general-purpose face recognition library with mobile applications. *CMU School of Computer Science* 6, 2 (2016), 20.

[16] Terry Anderson. 2008. *The theory and practice of online learning*. Athabasca University Press.

[17] Eric Arazo, Diego Ortego, Paul Albert, Noel E. O'Connor, and Kevin McGuinness. 2019. Unsupervised Label Noise Modeling and Loss Correction. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 312–321. http://proceedings.mlr.press/v97/arazo19a.html

[18] Tien-Duy B. Le, David Lo, Claire Le Goues, and Lars Grunske. 2016. A learning-to-rank based fault localization approach using likely invariants. In *Proceedings of the 25th international symposium on software testing and analysis*. 177–188.

[19] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. 2013. Better mixing via deep representations. In *International conference on machine learning*. PMLR, 552–560.

[20] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[21] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2018. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728* (2018).

[22] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. 2017. EMNIST: Extending MNIST to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2921–2926.

[23] Forrester Consulting. 2020. Overcome Obstacles To Get To AI At Scale. https://www.ibm.com/downloads/cas/VBMPEQLN.

[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[25] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. 2019. Deepfault: Fault localization for deep neural networks. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 171–191.

[26] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. 2017. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410* (2017).

[27] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 177–188.

[28] Benoît Frénay and Michel Verleysen. 2013. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems* 25, 5 (2013), 845–869.

[29] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*. PMLR, 1050–1059.

[30] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 3354–3361.

[31] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks* 18, 5-6 (2005), 602–610.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[33] Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).

[34] Thomas P Hettmansperger and Joseph W McKean. 2010. *Robust nonparametric statistical methods*. CRC Press.

[35] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[36] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1110–1121.

[37] Panagiotis G. Ipeirotis, Foster J. Provost, and Jing Wang. 2010. Quality management on Amazon Mechanical Turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP '10, Washington DC, USA, July 25, 2010*. ACM, 64–67. https://doi.org/10.1145/1837885.1837906

[38] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. 2018. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 2309–2318. http://proceedings.mlr.press/v80/jiang18c.html

[39] Youngdong Kim, Junho Yim, Juseung Yun, and Junmo Kim. 2019. Nlnl: Negative learning for noisy labels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 101–110.

[40] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[41] Shuming Kong, Yanyan Shen, and Linpeng Huang. 2021. Resolving training biases via influence-based data relabeling. In *International Conference on Learning Representations*.

[42] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. 2010. Outlier detection techniques. *Tutorial at KDD* 10 (2010), 1–76.

[43] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[44] Florian Laws, Christian Scheible, and Hinrich Schütze. 2011. Active learning with amazon mechanical turk. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. 1546–1556.

[45] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.

[46] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[47] Jichang Li, Guanbin Li, Feng Liu, and Yizhou Yu. 2022. Neighborhood Collective Estimation for Noisy Label Identification and Correction. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV*. Springer, 128–145.

[48] Junnan Li, Richard Socher, and Steven CH Hoi. 2020. Dividemix: Learning with noisy labels as semi-supervised learning. *arXiv preprint arXiv:2002.07394* (2020).

[49] Shikun Li, Xiaobo Xia, Hansong Zhang, Yibing Zhan, Shiming Ge, and Tongliang Liu. 2022. Estimating noise transition matrix with label correlations for noisy multi-label learning. In *Advances in Neural Information Processing Systems*.

[50] Xia Li and Lingming Zhang. 2017. Transforming programs and tests in tandem for fault localization. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–30.

[51] Yu Li, Min Li, Qiuxia Lai, Yannan Liu, and Qiang Xu. 2021. TestRank: Bringing Order into Unlabeled Test Instances for Deep Learning Tasks. *Advances in Neural Information Processing Systems* 34 (2021), 20874–20886.

[52] Yanhui Li, Linghan Meng, Lin Chen, Li Yu, Di Wu, Yuming Zhou, and Baowen Xu. 2022. Training Data Debugging for the Fairness of Machine Learning Software. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 2215–2227.

[53] Weixin Liang, Girmaw Abebe Tadesse, Daniel Ho, Fei-Fei Li, Matei Zaharia, Ce Zhang, and James Zou. 2022. Advances, challenges and opportunities in creating data for trustworthy AI. *Nature Machine Intelligence* (2022), 1–9.

[54] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common

objects in context. In *European conference on computer vision*. Springer, 740–755.
[55] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. 2017. A survey on deep learning in medical image analysis. *Medical image analysis* 42 (2017), 60–88.
[56] Zixi Liu, Yang Feng, and Zhenyu Chen. 2021. DialTest: automated testing for recurrent-neural-network-driven dialogue systems. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 115–126.
[57] Zixi Liu, Yang Feng, Yining Yin, and Zhenyu Chen. 2022. DeepState: Selecting Test Suites to Enhance the Robustness of Recurrent Neural Networks. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 598–609. https://doi.org/10.1145/3510003.3510231
[58] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering* 31, 12 (2018), 2346–2363.
[59] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 120–131.
[60] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test selection for deep learning systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 2 (2021), 1–22.
[61] Don McNicol. 2005. *A primer of signal detection theory*. Psychology Press.
[62] Seokhyeon Moon, Yunho Kim, Moonzoo Kim, and Shin Yoo. 2014. Ask the Mutants: Mutating Faulty Programs for Fault Localization. In *Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014, March 31 2014-April 4, 2014, Cleveland, Ohio, USA*. IEEE Computer Society, 153–162. https://doi.org/10.1109/ICST.2014.28
[63] Jaechang Nam, Wei Fu, Sunghun Kim, Tim Menzies, and Lin Tan. 2017. Heterogeneous defect prediction. *IEEE Transactions on Software Engineering* 44, 9 (2017), 874–896.
[64] Curtis Northcutt, Lu Jiang, and Isaac Chuang. 2021. Confident learning: Estimating uncertainty in dataset labels. *Journal of Artificial Intelligence Research* 70 (2021), 1373–1411.
[65] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. 2021. Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks. 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/f2217062e9a397a1dca429e7d70bc6ca-Abstract-round1.html
[66] Mike Papadakis and Yves Le Traon. 2015. Metallaxis-FL: mutation-based fault localization. *Softw. Test. Verification Reliab.* 25, 5-7 (2015), 605–628. https://doi.org/10.1002/stvr.1509
[67] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D Ernst, Deric Pang, and Benjamin Keller. 2017. Evaluating and improving fault localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 609–620.
[68] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.
[69] Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. 2019. CRADLE: cross-backend validation to detect and localize bugs in deep learning libraries. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1027–1038.
[70] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, Jeff Skowronek, and Linda Devine. 2006. Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen'sd indices the most appropriate choices. In *annual meeting of the Southern Association for Institutional Research*. Citeseer, 1–51.
[71] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.
[72] Weijun Shen, Yanhui Li, Lin Chen, Yuanlei Han, Yuming Zhou, and Baowen Xu. 2020. Multiple-boundary clustering and prioritization to promote neural network retraining. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 410–422.
[73] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[74] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. 2022. Learning from noisy labels with deep neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems* (2022).
[75] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. 2012. *Optimization for machine learning*. Mit Press.
[76] Bing Sun, Jun Sun, Long H Pham, and Jie Shi. 2022. Causality-based neural network repair. In *Proceedings of the 44th International Conference on Software Engineering*. 338–349.
[77] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.
[78] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. 2020. From imagenet to image classification: Contextualizing progress on benchmarks. In *International Conference on Machine Learning*. PMLR, 9625–9635.
[79] Zan Wang, Ming Yan, Junjie Chen, Shuang Liu, and Dongdi Zhang. 2020. Deep learning library testing via effective model generation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 788–799.
[80] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing test inputs for deep neural networks via mutation analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 397–409.
[81] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. Deeplocalize: Fault localization for deep neural networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 251–262.
[82] Anjiang Wei, Yinlin Deng, Chenyuan Yang, and Lingming Zhang. 2022. Free Lunch for Testing: Fuzzing Deep-Learning Libraries from Open Source. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 995–1007. https://doi.org/10.1145/3510003.3510041
[83] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.
[84] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. 2023. Deep isolation forest for anomaly detection. *IEEE Transactions on Knowledge and Data Engineering* (2023).
[85] Jifeng Xuan and Martin Monperrus. 2014. Learning to Combine Multiple Ranking Metrics for Fault Localization. In *2014 IEEE International Conference on Software Maintenance and Evolution*. 191–200. https://doi.org/10.1109/ICSME.2014.41
[86] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor Tsang, and Masashi Sugiyama. 2019. How does disagreement help generalization against label corruption?. In *International Conference on Machine Learning*. PMLR, 7164–7173.
[87] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor W. Tsang, and Masashi Sugiyama. 2019. How does Disagreement Help Generalization against Label Corruption?. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 7164–7173. http://proceedings.mlr.press/v97/yu19b.html
[88] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. 2020. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access* 8 (2020), 58443–58469.
[89] Jianxin Zhang, Yutong Wang, and Clay Scott. 2022. Learning from label proportions by learning with label noise. *Advances in Neural Information Processing Systems* 35 (2022), 26933–26942.
[90] Xiaoyu Zhang, Juan Zhai, Shiqing Ma, and Chao Shen. 2021. AUTOTRAINER: An Automatic DNN Training Problem Detection and Repair System. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 359–371.
[91] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *NIPS*.
[92] Xuezhou Zhang, Xiaojin Zhu, and Stephen Wright. 2018. Training set debugging using trusted items. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
[93] Ziqi Zhang, Yuanchun Li, Jindong Wang, Bingyan Liu, Ding Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2022. ReMoS: Reducing Defect Inheritance in Transfer Learning via Relevant Model Slicing. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 1856–1868.
[94] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. 2014. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*. Springer, 94–108.
[95] Yue Zhao, Zain Nasrullah, and Zheng Li. 2019. Pyod: A python toolbox for scalable outlier detection. *arXiv preprint arXiv:1901.01588* (2019).
[96] Zhaowei Zhu, Zihao Dong, and Yang Liu. 2022. Detecting corrupted labels without training a model to predict. In *International Conference on Machine Learning*. PMLR, 27412–27427.

Received 2023-02-02; accepted 2023-07-27