PhD-FSTM-2023-134
Faculty of Science, Technology and Medicine

Dissertation

Presented on the 14/12/2023 in Luxembourg

to obtain the degree of

Docteur de l'Université du Luxembourg en Informatique

by

Qiang Hu
Born on 27th May 1995 in Sichuan, China

Label-Efficient Deep Learning Engineering

Dissertation Defense Committee

Dr. Yves Le Traon, Member (Supervisor)
Full Professor, University of Luxembourg, Luxembourg

Dr. Tegawendé Bissyandé, Chair
Associate Professor, University of Luxembourg, Luxembourg

Dr. Maxime Cordy, Vice-chair
Research Scientist, University of Luxembourg, Luxembourg

Dr. Paolo Tonella, Member
Full Professor, Università della Svizzera Italiana, Switzerland

Dr. Lei Ma, Member
Associate Professor, The University of Tokyo, Japan

Abstract

Applying deep learning (DL) to science is a new trend in recent years which leads DL engineering to become an important problem. The normal processes of DL engineering contain data preparation, model architecture design, model training and testing, model deployment, and model maintenance. Unfortunately, all of them are complex and costly. One of the factors that highly affect the efficiency of DL engineering is the huge labeling effort in the data preparation process. High-quality labeled data can benefit the quality of the trained model and the reliability of model testing. However, even though collecting unlabeled raw data can be fully automated, labeling them needs huge human effort. Thus, how to efficiently build deep learning systems with less human effort (label-efficient DL engineering) is a key question we want to answer.

To tackle this label-efficient DL engineering problem, this dissertation studies existing methods then proposes new solutions and makes the following contributions.

- *Label-free model selection for efficient model design.* Since there are many open-sourced DNN models released every day, the good practice for developers to alleviate the effort of model design is to reuse the available model. We propose LaF, a label-free model selection method based on the Bayesian model to infer the models' specialty only based on predicted labels.
- *Label-efficient model training via active learning.* Active learning is already a mature learning framework to handle the labeling effort in model training. However, from a more practical view, how can existing active learning help us build a robust model is unknown. Here, robust means the adversarial robustness and the performance of the model after compression. We conduct an empirical study to explore these problems and reveal the limitations of existing active learning methods. Besides, we build the first benchmark of active learning for code models – active code learning to help software engineers train their code models with less effort.
- *Label-free model evaluation.* The traditional model testing process also needs labels for the prediction correctness checking. We propose a label-free model performance estimation method, Aries, to automate the entire testing process. Aries relies on the assumption that the model should have a similar prediction accuracy on the data which have similar distances to the decision boundary.
- *Label-efficient model enhancement.* The final process, model maintenance which is generally based on model retraining still needs the labeled data. We found existing Label-efficient model retraining methods suffer from the data distribution shift. Thus, we first conduct a systemically empirical study to reveal the impact of the retraining process and data distribution on model enhancement. Then based on our findings, we propose a novel distribution-aware test (DAT) selection metric that can produce a model robust model

against distribution shift.

To sum up, this dissertation shows promising ways to efficiently build a DL system with acceptable labeling effort that run through the entire DL engineering process from the model selection to model maintenance.

To my parents.

Acknowledgements

This dissertation would not have been possible without the support of many people who in one way or another have contributed and extended their precious knowledge and experience to my PhD studies. It is my pleasure to express my gratitude to them.

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Yves Le Traon, who has given me this great opportunity to come across continents to pursue my doctoral degree. He has always trusted and supported me with his great kindness throughout my whole PhD journey

Second, I am equally grateful to my co-supervisors, Dr. Maxime Cordy and Prof. Mike Papadakis, for their patience, advice, training, guidance, and encouragement. They taught me how to conduct research and present it to others. The discussions we had allowed me to better understand the field.

Third, I would like to extend my thanks to all my co-authors who worked closely with me during my PhD study including Dr. Yuejun Guo, Prof. Xiaofei Xie, Prof. Lei Ma, Dr. Wei Ma, Zeming Dong, Prof. Jianjun Zhao, etc.

I would like to thank all the members of my Ph.D. defense committee, including Prof. Tegawendé Bissyandé, Dr. Maxime Cordy, Prof. Paolo Tonella, Prof. Lei Ma, and my supervisor Prof. Yves Le Traon. It is my great honor to have them on my defense committee, and I appreciate very much their efforts to review my dissertation and evaluate my PhD work.

I would like to express my gratitude to all my colleagues from SERVAL (SnT) for all the good discussions we had and the interesting reading group sessions. Especially, thanks to my colleagues in the E008 office, Salah, Martin, and Jin, who provided me with a good work environment.

Besides, I would like to express my great thanks to my best friends in China, Xingyu Tao and Laixi Huang. Special thanks to Xingyu Tao who witnessed all my ups and downs since I was a child. I also want to thank my land owner Pengqian who cooked for me and offered me a lot of free beers. Thanks to Yuanyuan Yang for her support in the last period of my PhD.

More personally, I thank my parents. My parents support all my decisions and encourage me to push my dream bravely. I cannot be here without their unconditional love. Lastly, sincere thanks to my grandmother.

Qiang Hu
University of Luxembourg
September 2023

Contents

List of Figures

List of Tables

1 Introduction

This chapter introduces the concept of deep learning engineering first and then discusses the challenges in deep learning engineering. Finally, it summarizes the contributions and structure of this thesis.

Contents

1.1 Concept of Deep Learning Engineering

Deep learning (DL) becomes important in our daily life, therefore, how to efficiently develop and deploy DL systems becomes a crucial problem. We call the processes of building DL systems – deep learning engineering (DLE), also famous as Machine Learning Operations (MLOps). Figure 1.1 presents the workflow of DLE which contains six main processes, ❶ data preparation, ❷ model design, ❸ model training, ❹ model testing, ❺ model deployment, and ❻ model maintenance.

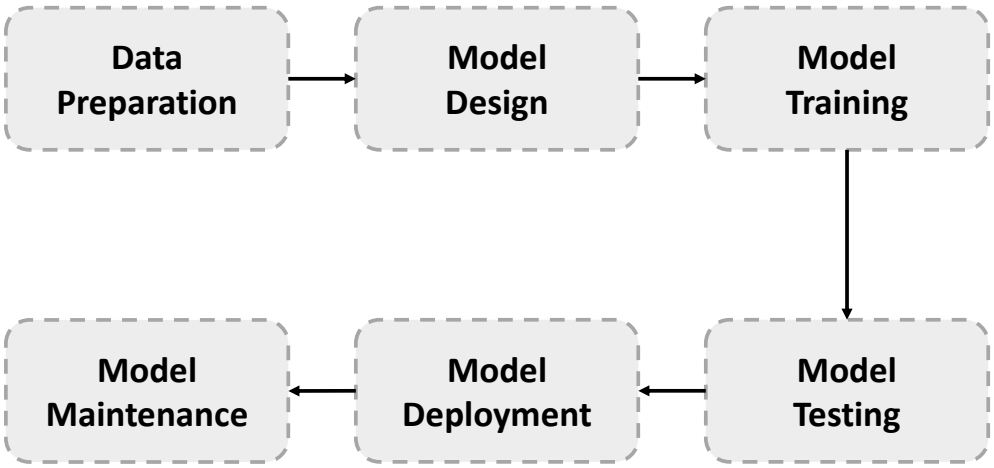


Figure 1.1: Overview of Deep Learning Engineering

Data preparation. Given a target task, e.g., building a digit classification model, the first step to building the DL system is to collect the related data. At the same time, data cleaning is also a necessary process to remove low-quality samples. After that, developers need to manually label the cleaned data to assign the target to each sample, i.e., which category the sample belongs to. Finally, the normal procedure is to randomly divide the collected data into three sets, training set, validation set, and test set. Normally, the training set works for optimizing the model parameters (e.g., model weights), and the validation set is useful for indicating the model with the best performance (e.g., test accuracy for classification model) in the training process. After we obtain a DNN model, developers can assess the model's performance on the unseen data (not included in the training data) using the test set.

Model design. Given the task and datasets prepared from the last step, developers need to design the architecture (e.g., layers, optimization functions) of the DNN model that is proper for the given task. Normally, DL developers will borrow some famous architectures as the backbone of their own models, e.g., use the ResNet [1] for image classification or use the transformer [2] for natural language processing.

Model training. After obtaining the dataset and model architecture, developers can train the model to tune its parameters using the training set. Model training is an automated process that does not need human effort. When the model performance evaluated on the validation set does not increase anymore or the training epoch is greater than the predefined number, the training process is finished and the trained model is under test.

Model testing. Before deploying the trained model to applications, developers

need to test the model to check if it fits the expectations. The test set prepared by the first step is used for accessing the performance (clean performance) of the DNN model. In addition to clean performance, there are some other properties that the developers might be interested in depending on the requirements, e.g., inference speed, adversarial robustness, and fairness. Of course, if the performance of the model is unexpected, the developers have to redesign the model architecture or change the training strategy to train the model again.

Model deployment. If the developers get a satisfied model, the next step is to deploy it into the application and put the system in the wild for usage. There are two commonly used model deployment processes. The first one is to directly use the trained model and the second one is to compress the model and then use the optimized version in the application. The second process is usually used when the computation resource of the system is limited or the model is too large to be deployed.

Model maintenance. Since the requirements and the deployed environments could change from time to time, the DL system needs to be updated accordingly. The common way to maintain the DNN model is to retrain it following the new requirements or using the data collected from the new environments.

1.2 Challenges with Deep Learning Engineering

DLE is heavy work since it contains complicated procedures where multiple challenges exist. This thesis focuses on two main challenges, the huge labeling effort for preparing datasets and the distribution shift that will introduce unexpected performance drops for deployed DNN models.

1.2.1 Labeling Effort

In the supervised scenario, both model training and model testing processes need the labeled data. For the model training, the labels will guide the model to tune the parameters following the right direction (based on the gradient calculation). For model testing, the labels are the ground truth for identifying the correctness of model prediction. However, even though collecting raw data is not difficult since it can be totally automated, labeling data is time-consuming and labor-intensive. More importantly, for some complicated tasks, e.g., tasks related to source code learning such as code repair, and clone detection, domain knowledge is required to label the collected raw data. For example, labeling the codes from only four libraries takes experienced software developers more than 600 man-hours [3]. As a result, how to reduce the labeling effort is the biggest challenge for DLE.

Challenges in label efficient model training. In the machine learning community, researchers already made efforts to alleviate the labeling effort for model training and proposed the famous learning framework – active learning. However, such efforts mainly lie in proposing new active learning methods to achieve state-of-the-art (SOTA) performance but ignore the problems in the real usage of active learning methods. For example, the existing works only evaluated the performance of active learning methods in terms of the clean accuracy on the test data of trained models, other properties such as the robustness of models are unexplored which raises the question of which active learning method we should use to produce a robust DNN model? Besides, existing active learning works focus on image classification and natural language process tasks, how can we use these methods to solve other

tasks, for example, software engineering tasks, is still unknown. How to propose new active learning methods for software engineering tasks that have different types of data with images and text is also unknown and needs to be explored.

Challenges in label efficient model testing. Different from model training, label-efficient model testing attracted limited attention. Existing label-efficient testing techniques can be divided into two groups, detect more faults in the unlabeled data pool under the fixed labeling budget, and label-efficient model evaluation. For the first type of technique, as revealed by [4], it is difficult to distinguish normal inputs (faults) and inputs with low uncertainty but are incorrectly predicted (with high uncertainty but correctly predicted). There are two scenarios in label-efficient model evaluation, 1) evaluate a single model and 2) evaluate multiple models. When we face a single model, we need to select a set to represent the whole set of the unlabeled set and test the model accordingly. How to define the *representative* is a difficult problem. When we need to evaluate multiple models and select the best one for the usage, the *representative* test set changes from representing one model to many models which brings a more challenging problem. Therefore, it is nice to try to avoid defining the *representative* data and just use the unlabeled data to assess the performance of models.

Challenges in label efficient model maintenance. The DNN model needs to be updated to fit new requirements or environments by mode retraining. However, if we label all the data every time, the labeling cost will be extremely high. Different from active learning which starts from a model with poor performance, retraining starts from a pre-trained model with good performance on a specific test set (the original test set). Therefore, the existing active learning methods can not be the best choice for this task. How to design the retraining method for a given environment is unknown and challenging.

1.2.2 Distribution Shift

An essential challenge in the realm of Deep Neural Network (DNN) testing and maintenance pertains to the phenomenon of distribution shift. Once a Deep Learning (DL) system has been deployed in real-world scenarios, the distribution of freshly acquired test data often diverges from the initial test data employed for performance assessment. Consequently, the established understanding of the DNN's performance becomes unreliable. Effectively evaluating the performance of DNN models on data afflicted by distribution shift presents a formidable obstacle. A similar predicament may manifest during the maintenance phase, wherein adaptation of the model to accommodate the altered data distribution resulting from distribution shift poses a significant challenge.

1.3 Contributions

This dissertation makes the following contributions:

- **A novel method to select DNN models without labeling effort.** We propose a novel label-free model selection technique, LaF, aimed at identifying the optimal-performing model without necessitating any manual labeling efforts. The principle underlying LaF revolves around the development of a Bayesian model, which seamlessly integrates both data complexity and model specialization to the estimation of the similarity between the model predictions and the ground-truth label. Note that, data complexity refers to the inherent

intricacy associated with achieving accurate predictions for diverse samples through various DNNs, as evidenced by the variability in predictions generated by multiple models. In contrast, model specialization assesses the efficacy of a given model in accurately deducing labels across the entire spectrum of samples, as manifested by the model's capacity to harmonize its predictions with the consensus among other models. By optimizing the Bayesian model, we can derive insights into the degree of model specialization, thereby providing valuable guidance for our model selection process. To assess the efficacy of our approach, we conducted comprehensive experiments utilizing 9 datasets and 165 DNNs. Besides, we consider both model accuracy and generalization. Experimental results demonstrated that LaF can rank models precisely and outperform our considered baselines with improvements of up to 0.53 Kendall's τ.

This work has been accepted by the ACM Transactions on Software Engineering and Methodology (TOSEM) in 2023.

- **An exploration of the limitation of active learning.** We address the following aspects and conduct empirical studies: 1) An exploration of the impact of active learning on the adversarial resilience of deep neural network (DNN) models. 2) A meticulous assessment of the inherent ability of models trained via active learning to uphold their accuracy in the wake of model compression, shedding light on the resilience of these models to the challenges of post-compression scenarios. 3) An in-depth investigation into the relationship between the quantity of training data, relationships entwining the quantity of training data, the models' capacity to withstand adversarial attacks, and the sustainability of their accuracy in the aftermath of model compression. In total, we trained more than 70000 DNN models for the study. We found that, while existing data selection metrics can generate accurate models, they occasionally introduce a compromise in the models' resistance to adversarial examples and their ability to withstand compression. In essence, our research underscores the presence of a delicate balance between the exertion of labeling efforts and the diverse qualities exhibited by different models. These findings light the future research, encouraging the proposal of new data selection metrics that holistically consider multiple dimensions of model quality.

  This work has been published in the 36th IEEE/ACM Conference on Automated Software Engineering (ASE 2019).

- **A benchmark and new insights of active code learning.** We build the first benchmark to study the problem of active code learning – sample-efficient training of code models. Concretely, we adapt acquisition functions from current active learning works for code learning. Then, experiments are conducted to explore the usefulness of these acquisition functions in the domain of code-related data. Our findings show that the effectiveness of active code learning is influenced by feature selection where using model outputs as features can produce the best code models. Nevertheless, considering the specific task of code summarization, the performance of active code learning exhibits noticeable limitations. In fact, it leads to models exhibiting a substantial performance

gap of over 29.64% when measured against expected performance benchmarks. Moreover, we conduct an exploratory study to explore future directions of active code learning. We propose to consider evaluation metrics such as the BLEU score as the distance calculation method. The experimental results show that there is a correlation between our proposed distance methods and the performance of code models.

This work is currently under submission to an international peer-reviewed journal.

- **A novel method to assess the performance of DNN models with free.** We design a new label-free method, *Aries*, to assess the performance of DNN models on unlabeled test sets. Our method capitalizes on the underlying insight that a correlation often exists between the accuracy of data predictions and their proximity to the decision boundary. That means two datasets that have similar distances to the decision boundary could have a similar prediction accuracy. We conduct in-depth evaluations to check the effectiveness of *Aries*. Furthermore, we broaden our investigation by incorporating 13 variations of transformed test sets, designed to simulate novel unlabeled data scenarios that might arise in real-world contexts. The results demonstrate that *Aries* estimate the accuracy with a difference ranging from 0.03% to 2.60% compared to the real accuracy. Remarkably, in comparison to state-of-the-art label-free methods [5], *Aries* demonstrates the capacity to provide more precise accuracy predictions. Notably, compared to sampling-based methods CES [6] and PACE [7], *Aries* is able to produce similar estimated results.

  This work has been published in the 45th International Conference on Software Engineering (ICSE 2023).

- **A study of distribution aware test selection and a novel data selection method for model retraining.** We found that existing sampling-based model retraining methods involve three significant limitations. 1) The existing methods diverge in terms of the retraining procedures they employ. 2) An oversight prevalent in these methods is the disregard for shifts in data distribution. 3) The evaluation of these methods is often insufficient. To fill this gap, we empirically explore the influence of diverse retraining processes and the variances introduced by distinct data distributions on the practice of retraining-based model enhancement. Our empirical findings reveal that combining the training and new data to retrain models can yield superior results compared to utilizing solely selected data. Leveraging the insights gleaned from this investigation, we design a new metric that considers data distribution, DAT, for model retraining. Our proposed DAT metric is able to mitigate the adverse effects of distribution shifts during the retraining process.

  This work has been published in the ACM Transactions on Software Engineering and Methodology (TOSEM) in 2022.

1.4 Organisation of the Dissertation

Figure 1.2 illustrates the roadmap of this dissertation. Chapter 2 introduces the necessary background information including deep learning, active learning, test optimization in DNN testing, and distribution shift. Chapter 3 discusses the existing works related to the contributions of this dissertation. Chapter 4 presents our label-free model selection method, LaF. Chapter 5 presents our empirical study on exploring the limitations of active learning. Chapter 6 introduces the benchmark we built for active code learning. Chapter 7 presents our proposed label-free model evaluation method, Aries. Chapter 8 introduces our empirical study on sampling-based model retraining and the new distribution-aware test selection method (DAT). Finally, Chapter 9 concludes this dissertation and discusses some potential future works.

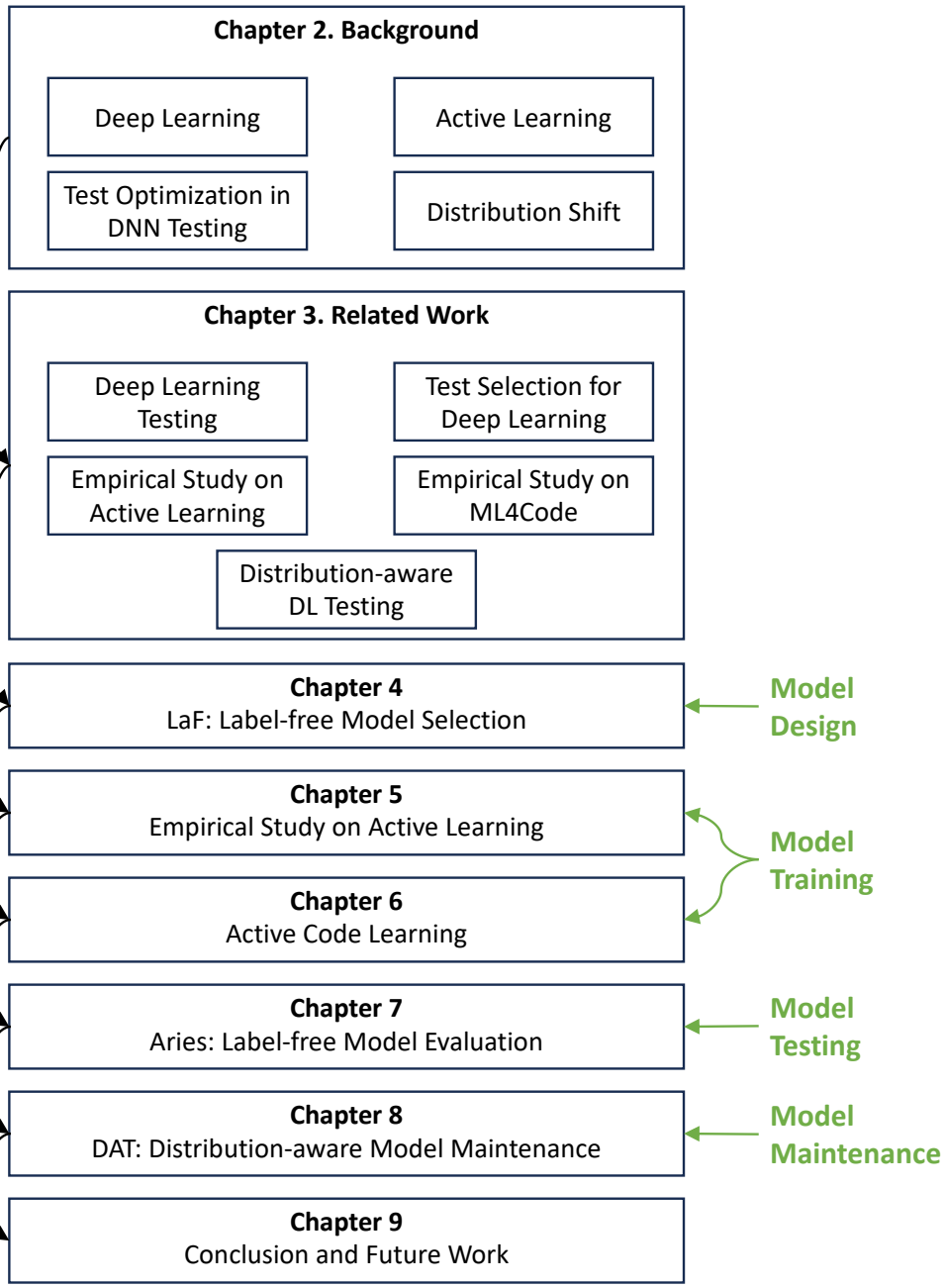


Figure 1.2: Roadmap of this dissertation.

2 Background

This chapter introduces the necessary knowledge to understand this dissertation. We revisit the details of deep learning, active learning, test optimization in DNN testing, and distribution shift.

Contents

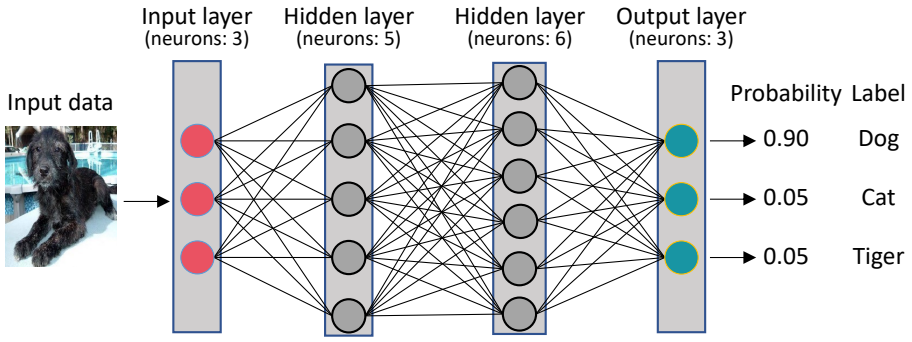


Figure 2.1: An example of a DNN (feed-forward neural networks) classifier [8].

2.1 Deep Learning

Conventional deep neural network (DNN) architectures can be typically characterized by multiple layers, encompassing an input layer, several hidden layers (optional), and an output layer. This structure is depicted in Figure 2.1, where each layer comprises numerous neurons represented as color circles. These neurons, also referred to as units or nodes, serve as the fundamental computational entities within a DNN. They gather information from input data or other neurons, utilizing activation functions to compute corresponding outputs. As illustrated in Figure 2.1, the input data is predicted to belong to the *Dog* with a probability of 0.90. The essence of DNN training revolves around the optimization of model parameters to minimize prediction errors concerning actual labels.

In the broader landscape of DNNs, there are two categories, Feed-Forward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs). FNNs are recognized for their unidirectional information flow, moving from the input layer to the output layer. They find extensive applications in image-processing domains. In contrast, RNNs adopt a distinct approach by incorporating inter-unit interactions, involving control units and memory cells, to facilitate the backward propagation of transferred knowledge within an RNN layer. This mechanism empowers RNNs to capture temporal data patterns effectively, making them particularly well-suited for sequential data processing tasks.

2.2 Active Learning

Active learning stands as a widely recognized strategy for achieving efficient model training with limited training samples. Figure 2.2 illustrates the active learning workflow. When we have an unlabeled dataset and a designed DNN architecture, The initial step is the selection of suitable features for data sampling. Generally, there are two groups of features. The first group contains the data features itself such as the image pixels and code tokens. The second group contains features extracted from the model, for example, code embeddings and model output. After feature extraction, acquisition functions identify the data instances with the highest representativeness, indicating their capacity to facilitate substantial improvements in model performance when labeled and integrated into the training dataset.

The acquisition function assumes paramount importance within active learning, as it is the decisive factor governing high-quality training data to label. Existing acquisition functions are broadly classified into two categories: those centered on

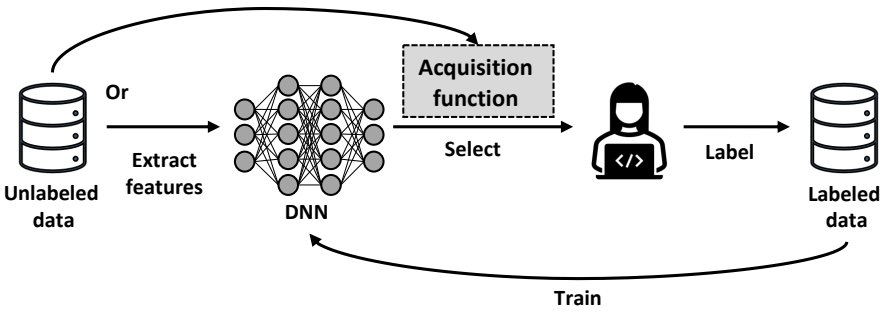


Figure 2.2: Overview of active learning [9].

the uncertainty of model outputs and those founded on feature clustering. Output-uncertainty-based functions begin by soliciting the model's predictions for the unlabeled data, subsequently employing uncertainty metrics, often measured through metrics like the variance of output probabilities, to rank these data. The intuition behind this type of function lies in the notion that data that the model has less confidence in are frequently situated close to the model's decision boundaries. Training DNN models with these uncertain data points serves to enforce the delineation of clear decision boundaries, thereby augmenting the model's accuracy and robustness. Clustering-based functions assume that models can attain improved performance by incorporating diverse data information into their training dataset. To achieve this, various distance metrics are employed to quantify data proximity, followed by the selection of central data points for model training. For instance, techniques like K-means clustering are employed to group data and subsequently select central points for training.

2.3 Test Optimization in DNN Testing

The conventional approach for assessing the accuracy of a Deep Neural Network (DNN) involves the manual labeling of each data point from unlabeled sets, and then computing the gap between truth labels and the model's predictions. However, this labeling process can be fraught with resource-intensive implications, encompassing both substantial labor costs and a significant time investment. As highlighted in existing literature, the concept of test selection [7] has emerged as a promising avenue for optimizing the testing phase. Test selection in DL represents a methodological framework designed to address two prevalent challenges. Firstly, it tackles the task of identifying a subset of data that is sufficiently representative of the entire dataset, enabling the estimation of the model's performance based on this selected subset. Secondly, it chooses data that are more prone to misclassification by the model, and then utilizes these data for retraining in order to enhance model accuracy.

2.4 Distribution Shift

Distribution shift stands as a pivotal concern within the domain of machine learning, signifying a scenario in which the data distributions characterizing training and test datasets diverge markedly. Specifically, when confronted with data afflicted by distribution shift, machine learning models typically experience heightened difficulty, rendering the reported performance based on in-distribution data useless.

Two categories of distribution shifts exist, namely synthetic distribution shift and

(a) White noise (b) Brightness (c) Location 1 (d) Location 2

Figure 2.3: Image examples with artificial distribution shifts (noise, brightness, location change). Images come from [12].

natural distribution shift. Synthetic distribution shift primarily materializes through the deliberate introduction of machine-generated perturbations or corruptions into the raw data. Figure 2.3 provides examples of synthetic distribution shift, showcasing the original image of a bird afflicted by different types of noise. Natural distribution shifts are typically injected by variations in environmental conditions. For instance, alterations in camera traps [10], new customer profiles [11], or newly introduced repositories [12] can precipitate a natural distribution shift. It is imperative to note that natural distribution shift exclusively manifests in data that has been collected and does not undergo deliberate perturbations. A recent benchmark [12] has been developed to encompass multiple natural distribution shifts.

3 Related Work

This chapter discusses related works from the perspectives of DNN testing, test selection for deep learning, empirical study on active learning, empirical study on ML4Code, and distribution-aware deep learning testing.

Contents

3.1 DNN Testing

Testing deep neural networks means the comprehensive evaluation of the quality of DNNs in anticipation of their subsequent deployment [13, 14, 15]. A simple and localized testing approach entails the partitioning of a dataset into training, validation, and test sets. The training and validation subsets are actively involved in the model's parameter optimization process, while the test set remains untouched and is used to measure the accuracy of models.

In contrast to conventional performance assessment, recent years have witnessed the introduction of numerous sophisticated testing techniques. Pei *et al.,* [16] proposed a neuron coverage-guided testing method, DeepXplore, for deep learning systems, inspired by the principles of code coverage-based testing in traditional software engineering. Coverage in this work is quantified based on the outputs of neurons in the DNN (e.g., if the output is greater than a threshold, the neuron is activated). Building upon this foundational testing criteria, Ma *et al.,* [17] further formulated various coverage criteria such as k-multisection Neuron Coverage. Utilizing coverage criteria, DeepTest [18] and TACTIC [19] have been devised to assess autonomous driving cars that based on DNNs. These two frameworks leverage coverage scores in the fitness functions to optimize search algorithms in the generation of test datasets designed to challenge the targeted systems by pinpointing potential vulnerabilities. Besides, the well-known technique of fuzzing testing has been adapted for the evaluation of deep learning models. The famous work [20] proposed a fuzzing testing framework, entailing the random introduction of noise into images to generate new test instances with the intent of identifying erroneous inputs in relation to DNN models. Xie *et al.,* [21] considered the seed selection problem in the fuzz process and further boosted the test generation for DNN systems. Guo *et al.,* [22] implemented DLFuzz, which is a framework that contains different fuzzing techniques, enriching the arsenal of advanced DL testing methodologies.

3.2 Test Selection for Deep Learning

Test selection in the context of DL targets to mitigate the human effort (e.g., labeling cost) associated with DL testing. Test selection has multiple goals. In this work, we focus on two representative goals, fault detection and model performance estimation.

Fault detection aims at identifying the mispredicted data before labeling them. In recent years, several methodologies have been proposed to facilitate this objective. For example, works [23, 24] introduced metrics grounded in the assessment of uncertainty inherent in output probabilities. They further demonstrated the utility of these metrics in the selective curation of data, subsequently harnessed for the retraining of pre-trained models, resulting in performance enhancements. Besides, Chen *et al.,* [25] proposed mutation testing-based fault detection. The basic idea in this technique is that faults are more likely be killed by mutation testing. Li *et al.,* [26] introduced a learning-based approach that leverages DNN models to learn the difference between the faults and normal data, culminating in the prediction of new fault instances. Gao *et al.,* [27] considered a multifaceted approach by considering the diversity of faults, thereby facilitating the selective curation of faults representing different fault patterns.

Performance estimation targeted methods aim at selecting a subset that users

can assess the model performance on the whole only based on the selected ones. Li *et al.,* [6] designed CES, a methodology selecting instances with the minimum cross-entropy concerning the whole dataset. Chen *et al.,* [28] introduce PACE, which employs clustering techniques to find the most representative samples, e.g., the sample at the center of each cluster. Guo *et al.,* [29] proposed the first performance estimation method, KAPE, for deep code search models. KAPE identifies the labeled training data that are similar to the unlabeled test data and then computes the performance of models accordingly.

3.3 Empirical Study on Active Learning

We witness the success of active learning in different scenarios. Various empirical investigations have been conducted across different domains. For instance, Ramirez-Loaiza *et al.* [30] conducted an empirical analysis of active learning methodologies, utilizing performance metrics such as accuracy, F1 score, and AUC for the evaluation of different baselines. Settles *et al.* [31] studied sequence labeling tasks by using active learning methods. Evaluation for these specific problem domains predominantly centered on learning curves and runtime. Yu *et al.* [32] conducted an empirical exploration of existing active learning techniques in the context of literature reviews. They subsequently introduced a novel approach that outperformed existing metrics with regard to different measurements. Chen *et al.* [33] studied active learning on word sense disambiguation problems, considering two fundamental active learning methods, entropy and margin, and assessing performance solely based on accuracy. Heilbron *et al.* [34] empirically explored the application of action localization tasks. However, they only considered three active learning methods. Their evaluation predominantly relied on the ALC metric, centered on classification correctness. Manabu [35] studied active learning within the domain of natural language processing, particularly with support vector machines (SVMs). Their study primarily compared the proposed metrics with random selection and concluded that the proposed method performs well on Japanese word segmentation.

3.4 Empirical Study on ML4Code

Machine learning for code (ML4Code) has garnered substantial attention in recent times, prompting extensive empirical investigations to elucidate the nuances and challenges within the ML4Code domain. In an early study by Chirkova *et al.* [36], researchers empirically examined the foundational model architecture of subsequent code models, specifically Transformers, across diverse code-related tasks encompassing code completion, function naming, and bug fixing. The study findings underscored the aptitude of Transformers to effectively leverage syntactic information to address code-related tasks. Niu *et al.* [37] conducted an empirical study to compare pre-trained models designed for source code across more than 10 tasks and models. Their study furnished detailed recommendations for the utilization and evaluation of these models, enhancing the understanding of their practical applications. Steenhoek *et al.* [38] studied DL models used for vulnerability detection. The results of their experiments demonstrated that models trained for specific types of vulnerabilities outperformed those trained for all vulnerabilities. Furthermore, the study revealed that increasing the size of training datasets yielded limited benefits. Hu *et al.* [39] contributed to the field by collecting and providing

distribution-shifted code datasets and investigating the generalization capabilities of code models under this dataset. The study findings illuminated the susceptibility of code models to distribution shifts, highlighting the challenges in addressing these shifts effectively. More recently, Nie *et al.* [40] explored the impact of using different evaluation methods on code models. Their findings underscored the existence of conflicting results arising from different evaluation methodologies, underscoring the need for heightened attention to evaluation processes during code model testing.

More recently, researchers employed large language models (LLM) to tackle code-related tasks, yielding impressive results in diverse areas [41, 42, 43]. Xia *et al.* [44] employed LLMs to address automated program repair challenges, achieving state-of-the-art outcomes within acceptable resource constraints. Deng *et al.* [45] leveraged LLMs to synthesize unconventional programs for testing deep learning libraries, unearthing 49 hitherto unknown bugs within prominent deep learning frameworks. Ma *et al.* [46] empirically scrutinized the capabilities of ChatGPT, one of the most renowned LLMs, in terms of program syntax, static analysis, and dynamic understanding. The results elucidated ChatGPT's proficiency in analyzing program syntax and static information while revealing a proclivity for generating non-existent semantic information.

3.5 Distribution-aware Deep Learning Testing

The negative impact of data distribution on the landscape of deep learning testing has been addressed recently, particularly within the test generation. David *et al.* [47] empirically investigated the intricate relationship between DL testing criteria (e.g., neuron coverage) and the data distribution. Their work not only shed light on this relationship but also offered invaluable research insights. One such insight underscores the imperative for deep learning testing tools to exhibit a heightened awareness of data distribution dynamics. Furthermore, Swaroopa *et al.* [48] designed a new test generation approach that considers the data distribution information, rooted in variational auto-encoders (VAE). Their research commenced with a meticulous examination of the veracity and realism of test data generated by extant test generation techniques, including notable methods such as DeepXplore. Subsequently, they devised an innovative test generation methodology characterized by a unique attribute. This method scrutinizes the authenticity and validity of the generated data at each juncture of the generation process, introducing a critical layer of quality control into the test data generation pipeline.

4 LaF: Labeling-Free Model Selection for Automated Deep Neural Network Reusing

In this chapter, we propose a labeling-free model selection method, LaF, to identify the model that has the best performance without any labeling effort. The core concept behind LaF involves the creation of a Bayesian model that integrates both data complexity and model specialization to gauge the probability that a predicted label corresponds to the actual label. Via optimizing the Bayesian model, we can deduce the degree of model specialization, which then guides our model selection process.

Contents

4.1 Introduction

Deep learning (DL) is helping solve all sorts of real-world problems in various domains, such as computer vision [49], natural language processing (NLP) [50], code understanding [51], and autonomous driving [52]. Due to the outstanding performance of deep neural networks (DNNs), software engineering (SE) researchers have attempted to apply DNNs to solve various SE tasks, such as source code processing [51, 53], automatic software testing [54, 55], and GUI designs [56]. Building machine learning (or deep learning) systems generally requires model architecture design and data preparation, model training, and model deployment and maintenance, known as machine learning operations (MLOps) [57]. However, since 1) designing new DNN architectures requires tremendous experimentation and DL knowledge, 2) preparing massive labeled training data is labor-intensive, and 3) training the DL model needs a considerable amount of time and resources, developers generally reuse generic and publicly available pre-trained models to solve their given task. Ultimately, we expect model reuse to become the norm in DL-based SE, just like it has been in other fields like NLP.

Due to the convenience of multiple public open sources such as GitHub [58], TensorFlow datasets [59], and Hugging Face [60], engineers can today access a massive number of DNNs, in the form of either pre-trained model files (e.g., .h5 and .pth). While this is practical, there is no prior evidence of which model will be capable of solving the targeted task the most effectively. Indeed, different models have been developed by different contributors and in different development settings, while they have been evaluated in unknown or incomparable testing conditions. For example, only 21 of the 165 DNNs we collected and evaluated in this paper have a performance report.

Model selection is a process of determining the best fit from multiple models for a given test set. Selecting candidate models for the targeted task raises two challenges. First, test data that can be found in the real world (e.g. source code from public repositories) are generally unlabelled [61], while data labeling requires significant manual effort. For example, the AIZU online programming challenge [62] receives submissions in different programming languages all the time. A Java developer can easily annotate the source code in Java but may have difficulties with other programming languages, such as C++. The effective selection of pre-trained models to solve various tasks and overcome the challenge of insufficient domain knowledge, therefore, requires a precise and efficient method to **compare and rank** DNN candidates **without** data labels.

An additional challenge originates from the fact that the chosen models will likely be confronted with data that have a different distribution from the data they were trained/tested on, i.e. out-of-distribution (OOD) data [63, 64, 48]. For many application domains – including software [12] – the distribution shift phenomena that yield OOD data are inevitable. As a result, these models may exhibit remarkably different performances over time, which raises the concern of quality and reliability [65]. For instance, for the same dataset iWildCam (please refer to Section 4.3.3 for more details), two DNNs exhibit 75.74% and 76.60% accuracy on the initial test data, while their performance turns to 76.82% and 65.30%, respectively, on OOD data. In conclusion, this suggests that ideal selection methods can handle the data distribution shift problem and reliably estimate model performance on OOD data.

Such methods have been proposed by a recent approach named sample discrimination-based selection (SDS) [66]. SDS achieves positive model ranking results on three benchmark datasets. SDS selects a set of data to label based on the majority voting [67] and item discrimination [68]. These data are considered the most discriminative in terms of distinguishing the accuracy between DNNs. DNNs are then ranked based on their accuracy on these selected and labeled data. Figure 4.1 illustrates how this sampling-based approach works. Although promising, SDS still suffers from manual labeling. Besides, it has only considered ranking based on model accuracy with in-distribution (ID) data (i.e. data that follow the same distribution as the data the models were trained on) – it has disregarded OOD data. Third, it has been applied to the image domain only; its effectiveness for other domains remains unclear.

In this paper, we aim to overcome the above limitations and propose LaF, a labeling-free model selection method, for DNNs that is effective with both ID and OOD data. Given a sample, only the predicted labels of multiple DNNs are available. The main idea of LaF is to build a Bayesian model that incorporates the data difficulty and model specialty to estimate the likelihood of a predicted label being the true label. The data difficulty implies how difficult a sample is for all DNNs to predict correctly, which is reflected by the prediction difference across multiple models. The model specialty indicates how good a model is to infer the correct labels of all samples, which is reflected by the ability to have the same predictions as the majority of models. Via optimizing the Bayesian model, we infer the model specialty to perform the model selection. The optimization is achieved by the expectation-maximization (EM) algorithm [69], which is efficient in finding maximum likelihood parameters (the data difficulty and model specialty in our case). To summarize, the main contributions of our work are:

1. We propose a novel approach, LaF, for automatically ranking multiple DNN models to facilitate the reuse of DNNs from public sources.

2. We demonstrate the effectiveness of LaF on ID and OOD data. Both the artificial and natural distribution shifts are considered.

3. LaF is labeling-free, which makes it practical and feasible in real-world applications.

4. We experiment on 9 benchmark datasets spanning different domains including image, text, and source code with different programming languages (Java and C++). To the best of our knowledge, this is the first DNN model selection work containing datasets other than images.

5. All the used models, datasets, and implementations of LaF and baseline methods are publicly available at `https://github.com/testing-cs/LaF-model-selection.git`.

4.2 Methodology

4.2.1 Problem Formulation

In this paper, we are interested in the classification task. Given a C-class task over a sample space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y} \to \mathcal{R}$, where $x \in \mathcal{X}$ is an input data and $y \in \mathcal{Y}$

is its class label. Let $f : x \to y$ be a deep neural network (DNN) that maps x to the problem domain. Given n models, $f_1, f_2, \ldots, f_n$, extracted from public sources and a set of unlabeled test data T, the problem we study is to estimate the rank of models regarding their performance on $\boldsymbol{T} = \{x_1, x_2, \ldots, x_m\}$. Figure 4.1 illustrates the workflow.

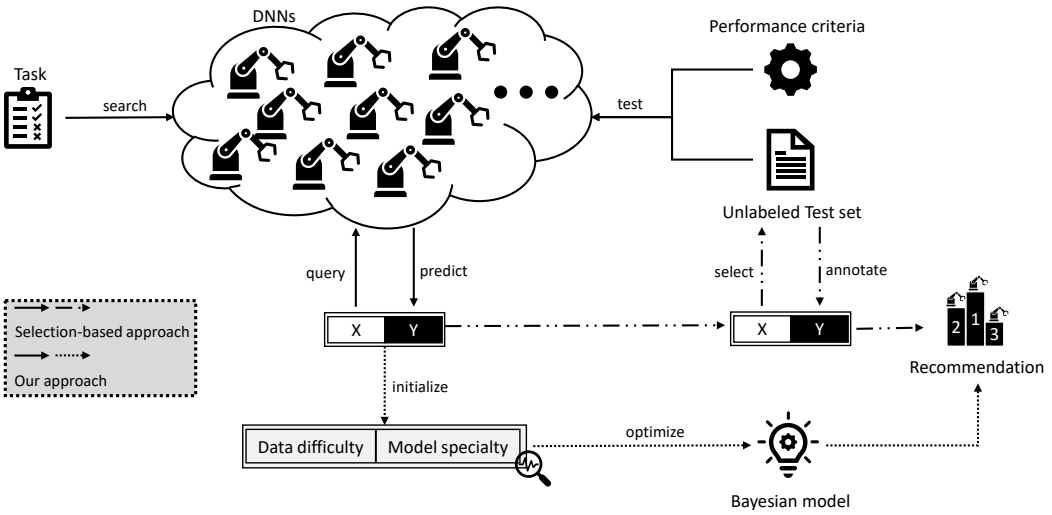


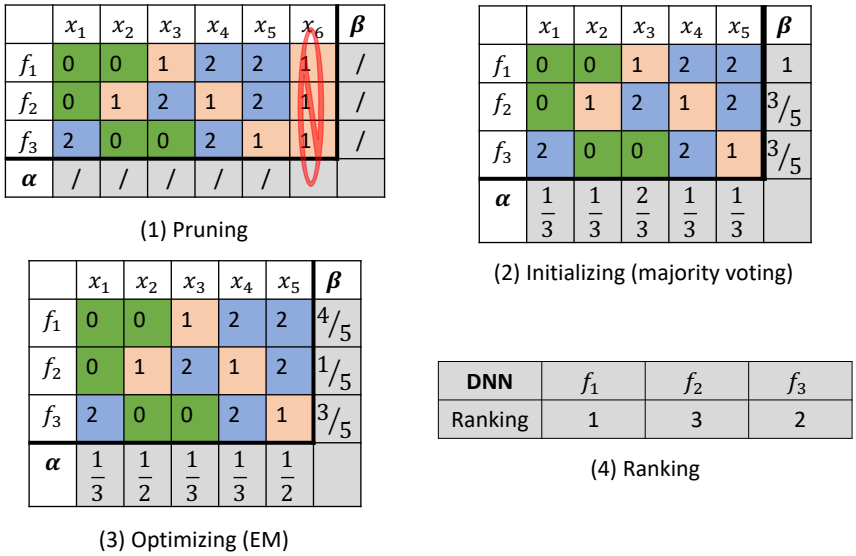
Figure 4.1: Illustration of the studied problem.

We assume that compared to the time cost of labeling data, the time cost of computation is negligible. For example, labeling code data from only four libraries can take up to 600 man-hours [70] whereas LaF and the competing methods produce a ranking with much less time (e.g., ranking Java250-based models with less than 15 minutes). To this end, we propose to tackle the ranking problem by only querying the predictions, which is highly applicable in practical scenarios. Remarkably, in this paper, we consider the performance of both accuracy and robustness. The accuracy is the correctness ratio of prediction on ID data. The robustness is the correctness ratio of OOD data.

4.2.2 Motivating Example

Figure 4.2 gives an example of LaF. In this simple 3-class example, there are 3 DNN models (f_1, f_2, f_3) given 6 unlabeled samples ($x_1, x_2, \ldots, x_6$). The goal is to rank the 3 models concerning their accuracy on these samples in the absence of true labels. First, we compute the predicted label of each sample by each model and remove x_6 where all the models have the same prediction. Second, we initialize the two parameters, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, contained in our approach. $\boldsymbol{\alpha}$ refers to how difficult a sample is for all models to predict the correct label. $\boldsymbol{\beta}$ indicates how good a model is to output the correct labels of all samples. Initially, we use the simplest and most commonly used majority voting heuristic [67] to give a pseudo label to each sample. For instance, the pseudo label of x_1 is 0 because 2 (f_1, f_2) of 3 models predict the label as 0. $\boldsymbol{\alpha}$ is defined as the ratio of mismatched models that output a different label instead of the pseudo one. $\boldsymbol{\beta}$ is calculated as the ratio of correctly predicted samples over the entire set. Third, since the pseudo labels are not the true labels, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ cannot truly reflect the data difficulty and model ability. We optimize these two parameters by a likelihood estimation method in the presence of true labels. Finally, based on the optimized $\boldsymbol{\beta}$ ($\frac{4}{5}, \frac{1}{5}, \frac{3}{5}$), we obtain the ranking 1, 3, and 2 for

f_1, f_2, and f_3, respectively.



	x_1	x_2	x_3	x_4	x_5	x_6	$\boldsymbol{\beta}$
f_1	0	0	1	2	2	1	/
f_2	0	1	2	1	2	1	/
f_3	2	0	0	2	1	1	/
$\boldsymbol{\alpha}$	/	/	/	/	/		

	x_1	x_2	x_3	x_4	x_5	$\boldsymbol{\beta}$
f_1	0	0	1	2	2	1
f_2	0	1	2	1	2	$3/5$
f_3	2	0	0	2	1	$3/5$
$\boldsymbol{\alpha}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	

	x_1	x_2	x_3	x_4	x_5	$\boldsymbol{\beta}$
f_1	0	0	1	2	2	$4/5$
f_2	0	1	2	1	2	$1/5$
f_3	2	0	0	2	1	$3/5$
$\boldsymbol{\alpha}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{2}$	

DNN	f_1	f_2	f_3
Ranking	1	3	2

Figure 4.2: An example of our four-step approach. It ranks 3 DNNs (f_1, f_2, f_3) given 6 unlabeled data (x_1, x_2, x_3, x_5, x_6) in a 3-class classification task. Numbers (0, 1, 2) highlighted in colors are predicted labels.

4.2.3 Our Approach: LaF

Given that no label is available in the test data, the main idea of our approach is to infer the specialties of DNNs by approximately maximizing the likelihood between the predictions and true labels via the expectation-maximization (EM) algorithm [69]. Let $\widetilde{\boldsymbol{Y}} = \{\widetilde{y}_{ij}\}_{1\le i\le m, 1\le j\le n}$ be the predicted labels of $\boldsymbol{T}$ and $\boldsymbol{Y} = \{y_i\}_{1\le i\le m}$ be the true labels. Here, $\widetilde{y}_{ij}$ refers to x_i and model f_j. Given the observed $\widetilde{\boldsymbol{Y}}$ and latent $\boldsymbol{Y}$ governed by unknown parameters $\boldsymbol{\theta}$, the likelihood function is defined as $L\left(\boldsymbol{\theta}; \widetilde{\boldsymbol{Y}}\right) = p\left(\widetilde{\boldsymbol{Y}} \mid \boldsymbol{\theta}\right) = \sum_{i=1}^{m} p\left(\widetilde{\boldsymbol{Y}}, y_i \mid \boldsymbol{\theta}\right)$. The goal is to search the best $\boldsymbol{\theta}$ that maximizes the likelihood, in other words, the probability of observing $\widetilde{\boldsymbol{Y}}$. As for $\boldsymbol{\theta}$, inspired by [71], we consider two factors, data difficulty $\boldsymbol{\alpha} = \{\alpha_i\}_{1\le i\le m}$ and model specialty $\boldsymbol{\beta} = \{\beta_j\}_{1\le j\le m}$, that influence the performance of DNNs. Namely, $\boldsymbol{\theta} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$. Algorithm 1 presents the pseudo-code of our approach.

Step 1: Pruning. Inevitably, some data will receive the same predictions by all models, which is useless for discriminating the performance and causes computational cost. For this reason, we filter these data without losing any information for ranking and obtain a smaller set $\boldsymbol{T'}$ (Lines 1-6 in Algorithm 1).

Step 2: Initializing. First, for each data x_i, since we do not have its ground-truth label, we use the voted label by DNNs as the replacement, namely, $y'_i = \text{mode}\left(\{\widetilde{y}_{ij}\}_{1\le j\le n}\right)$ (Lines 7-9). Next, α_i is the number of DNNs that gives a different label from the pseudo label and β_j is the accuracy based on pseudo labels (Lines 10-15). Formally, the definitions are:

$$\alpha_i = \frac{\{\widetilde{y}_{ij} \mid \widetilde{y}_{ij} \neq y'_i, 1 \le j \le n\}}{n}, \beta_j = \frac{\{\widetilde{y}_{ij} \mid \widetilde{y}_{ij} = y'_i, 1 \le i \le |\boldsymbol{T'}|\}}{|\boldsymbol{T'}|} \tag{4.1}$$

Step 3: Optimizing. To obtain the best results of the model parameters (data difficulty and model specialty), we use the EM algorithm [69] which has been proven

to be a powerful tool for estimating the parameters of statistical models to solve the optimization problem. Specifically, EM performs an expectation (E) step and a maximization (M) step (Lines 16-26) in the optimization process. In the E-step, it estimates the expected value of the log-likelihood:

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta_{last}}) &= \mathrm{E}\left[\log L\left(\boldsymbol{\theta}; \widetilde{\boldsymbol{Y}}, \boldsymbol{Y}\right)\right] \\ &= \sum_{i=1}^{|\boldsymbol{T'}|} \mathrm{E}\left[\log\left(p\left(y_i\right)\right)\right] + \sum_{i=1}^{|\boldsymbol{T'}|} \sum_{j=1}^{n} \mathrm{E}\left[p\left(\widetilde{y}_{ij} \mid y_i, \alpha_i, \beta_j\right)\right] \end{aligned} \tag{4.2}$$

where $\boldsymbol{\theta} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ and $\boldsymbol{\theta_{last}}$ is from the last E-step. For the computation, we use the definition from [71] where $p\left(\widetilde{y}_{ij} = y_i \mid \alpha_i, \beta_j\right) = \frac{1}{1+e^{-\alpha_i \beta_j}}$. Besides, as $\widetilde{y}_{ij}$ and $\boldsymbol{\beta}$ are independent given $\boldsymbol{\alpha}$, $p\left(y_i\right) = p\left(y_i \mid \alpha_i, \boldsymbol{\beta}\right)$. Remarkably, y_i represents the true label of a sample. In the ranking problem, y_i is absent but the probability of taking it as a true label can be inferred by $p\left(y_i \mid \alpha_i, \boldsymbol{\beta}\right)$.

In the M-step, the gradient ascent is applied to search for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ that maximize Q:

$$\boldsymbol{\theta_{new}} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta_{last}}) \tag{4.3}$$

where $\boldsymbol{\theta_{new}}$ is the updated parameters for the next iteration.

Step 4: Ranking. Finally, as $\boldsymbol{\beta}$ well estimate the abilities of each DNN given the observed labels, we use this vector to rank DNNs (Line 27). A high specialty indicates a good performance on the data.

Overall, LaF is a parameter-free method where the influence factors in LaF – the initialization of data difficulty and model specialty are automatically decided by the majority voting.

4.3 Experimental Setup

4.3.1 Implementation

All experiments were conducted on a high-performance computer cluster and each cluster node runs a 2.6 GHz Intel Xeon Gold 6132 CPU with an NVIDIA Tesla V100 16G SXM2 GPU. We implement the proposed approach and baseline methods based on the state-of-the-art frameworks, Tensorflow 2.3.0 and PyTorch 1.6.0. For the artificial data distribution shift, we consider 3 benchmark datasets where each includes 15 types of natural corruption with 5 levels of severity. In total, we test on $3 * 15 * 5 = 225$ datasets with the artificial distribution shift. We report the average results on corrupted data for baseline methods. The entire results corroborate our findings and are available on our companion project website [72].

4.3.2 Research Questions

In this study, we focus on the following three research questions:

- RQ1 (**effectiveness given ID test**): How is LaF ranking multiple DNNs given ID test data?
- RQ2 (**effectiveness under distribution shift**): How is LaF ranking multiple DNNs given OOD test data (including artificial and natural distribution shifts)?
- RQ3 (**ablation study**): How does each component contribute to LaF?
- RQ4 (**impact factors of LaF**): What is the impact of model quality, model diversity, model number, and input number on ranking DNNs?

Algorithm 1: LaF: Labeling-free comparison testing

Input : $\{f_1, f_2, \ldots, f_n\}$: DNNs for comparison
$\boldsymbol{T} = \{x_1, x_2, \ldots, x_m\}$: test set
Γ: performance criterion

Output : $\{r'(f_1), r'(f_2), \ldots, r'(f_n)\}$: Rank of DNNs

`/* Step1: Pruning */`

$\boldsymbol{T'} = \{\}$

for $i = 1 \rightarrow m$ **do**
  if $\left|\left\{\{\widetilde{y}_{ij}\}_{1\leq j\leq n}\right\}\right| > 1$ **then**
    $\boldsymbol{T'} \leftarrow x_i$; `//` $\widetilde{y}_{ij}$ `is the predicted label by` f_j
  end
end

`/* Step 2: Initializing */`

for $i = 1 \rightarrow |\boldsymbol{T'}|$ **do**
  $y'_i = \text{mode}\left(\{\widetilde{y}_{ij}\}_{1\leq j\leq n}\right)$; `// Majority voting`
end

for $i = 1 \rightarrow |\boldsymbol{T'}|$ **do**
  $\alpha_i = \frac{\{\widetilde{y}_{ij} | \widetilde{y}_{ij} \neq y'_i, 1\leq j\leq n\}}{n}$; `// Data difficulty`
end

for $j = 1 \rightarrow n$ **do**
  $\beta_j = \frac{\{\widetilde{y}_{ij} | \widetilde{y}_{ij} = y'_i, 1\leq i\leq |\boldsymbol{T'}|\}}{|\boldsymbol{T'}|}$; `// Model specialty`
end

`/* Step 3: Optimizing */`

$\boldsymbol{\alpha}_{last} = \{\alpha_i\}_{1\leq i\leq |\boldsymbol{T'}|}$

$\boldsymbol{\beta}_{last} = \{\beta_j\}_{1\leq j\leq n}$

$Q_{last} = computeQ(\boldsymbol{\alpha}_{last}, \boldsymbol{\beta}_{last})$; `//` *ComputeQ* `estimates the log likelihood based on Equation (4.2)`

$\boldsymbol{\alpha}, \boldsymbol{\beta} = gradientAscent(\boldsymbol{\alpha}_{last}, \boldsymbol{\beta}_{last}, Q_{last})$

$Q = computeQ(\boldsymbol{\alpha}, \boldsymbol{\beta})$

while $\left|\frac{Q - Q_{last}}{Q_{last}}\right| > 1E-5$ **do**
  $Q_{last} = Q$
  $\boldsymbol{\alpha}_{last}, \boldsymbol{\beta}_{last} = \boldsymbol{\alpha}, \boldsymbol{\beta}$
  $\boldsymbol{\alpha}, \boldsymbol{\beta} = gradientAscent(\boldsymbol{\alpha}_{last}, \boldsymbol{\beta}_{last}, Q_{last})$
  $Q = computeQ(\boldsymbol{\alpha}, \boldsymbol{\beta})$
end

`/* Step 4: Ranking */`

$r'(f_1), r'(f_2), \ldots, r'(f_n) = Sort(\boldsymbol{\beta})$

return $\{r'(f_1), r'(f_2), \ldots, r'(f_n)\}$

The first two research questions successively evaluate the effectiveness of our proposed solution given test data with and without distribution shift. The second one also tends to show how flexible and practical LaF is in real-world applications, especially by the test data with natural distribution shifts. The third research question studies the contribution of each component in LaF. The last one investigates the impact factors that may affect the ranking performance.

4.3.3 Datasets and DNNs

Datasets. We choose seven datasets, MNIST [73], Fashion-MNIST [74], CIFAR-10 [75], iWildCam [10], Amazon [11], Java250, and C++1000 [51] that are widely studied in previous work. These datasets cover the image (first 4), text (Amazon), and source code (Java250 and C++1000) domains. The test data that follow the same distribution as the training set are the so-called in-distribution (ID) data. The test data with data distribution shift are seem as out-of-distribution (OOD) data. In our work, we consider two types of distribution shifts: artificial and natural. For the artificial distribution shift, we use two benchmark datasets, MNIST-C [76] and CIFAR-10-C [77], for MNIST and CIFAR-10, respectively. We follow the official implementation of MNIST-C and select 15 types of corruption that are also used in CIFAR-10-C for our experiments, such as Gaussian noise, shot noise, impulse noise, defocus blur, frosted glass blur, motion blur, zoom blur, snow, frost, fog, brightness, contrast, elastic, pixelate, and jpeg. Besides, each type of corruption has 5 levels of severity. In total, our used MNIST-C and CIFAR-10 datasets contain 75 different test sets. However, there are no existing benchmark datasets for the distribution shift version of Fashion-MNIST, we simply follow the same setting of MNIST-C and use their script to generate Fashion-MNIST-C. For the natural distribution shift, we use two datasets for iWildCam and Amazon, respectively, from a recent-published benchmark, WILDS [12]. The distribution shift comes from new came traps in iWildCam and new users in Amazon. For Java250, we manually collect the OOD dataset based on the definition in WILDS that the distribution shift of source code comes from new repositories. For each class in Java250, we extract java files from [51] under the constraint that the corresponding users do not exist in ID data. Unfortunately, we did not find suitable OOD data for C++1000. Table 4.1 lists the details of datasets.

Table 4.1: Summary of datasets. "#ID" is the number of in-distribution test data. "#OOD" is the number of out-of-distribution test data with artificial or natural distribution shifts.

Dataset	Domain	Data Type	#Classes	#ID	#OOD	Distribution Shift
MNIST	Computer vision	Image of handwritten digits	10	10,000	750,000	Artificial
Fashion-MNIST	Computer vision	Image of fashion products	10	10,000	75,000	Artificial
CIFAR-10	Computer vision	Image of animals and vehicles	10	10,000	750,000	Artificial
iWildCam	Computer vision	Image of wildlife	182	8,154	42,791	Natural
Amazon	Natural language processing	Text of comments	5	46,950	100,050	Natural
Java250	Source code analysis	Source in Java	250	15,000	15,000	Natural
C++1000	Source code analysis	Source code in C++	1,000	99.997	-	-

DNNs. From Github, we collect, in total, 165 models, 30 for MNIST, 25 for Fashion-MNIST, 30 for CIFAR-10, 20 for iWildCam, 20 for Amazon, 20 for Java250, and 20 for C++1000. In concrete, the models of MNIST and CIFAR-10 are extracted using the GitHub links in [66]. The 25 models of Fashion-MNIST are extracted from [78] [66]. For iWildCam and Amazon, we train models using the implementation

in the benchmark WILDS [12]. For Java250 and C++1000, we train models using the implementation in the benchmark Project CodeNet [51] by different optimizers (sgd, rmsprop, adam, adadelta, adagrad, adamax, nadam, ftrl) and architectures (basic, doublePool). Table 4.2 presents the accuracy and robustness of DNNs on ID test data and OOD test data with natural distribution shift, respectively.

Table 4.2: Summary of models. "#DNNs" is the number of DNNs collected for each dataset. "#Parameters" shows the minimum and the maximum number of parameters of collected DNNs. "Accuracy" and "Robustness" lists the lowest and highest accuracy and robustness on test data with and without distribution shift, respectively. MNIST-C and CIFAR-10-C are two benchmark datasets and corresponding robustness is summarized in Table 4.12.

Dataset	#DNNs	#Parameters	Accuracy (%)	Robustness (%)
MNIST	30	7,206-3,274,634	85.27-99.54	MNIST-C
Fashion-MNIST	25	258,826-1,256,080	90.09-93.38	Fashion-MNIST-C
CIFAR-10	30	62,006-45,294,194	69.90-95.92	CIFAR-10-C
iWildCam	20	7,224,054-23,960,630	75.72-77.26	65.30-76.82
Amazon	20	3,223,255-12,861,655	56.06-59.13	38.24-56.94
Java250	20	2,927,290	64.73-87.39	57.24-81.67
C++1000	20	4,409,288	71.39-92.10	-

4.3.4 Baseline Methods

In our study, we compare LaF to four baseline methods, including one labeling-free method (CRC) and three sampling-based methods (random sampling, SDS, and CES). For conducting sampling-based methods, we set the labeling budgets from the number of DNNs (i.e., 30 DNNs in MNIST) to 180 at intervals of 5. Here, we follow the previous work [66] to set the maximum labeling budget as 180. Moreover, since our method is labeling-free, we tend to compare LaF with sampling-based methods that use as little labeling budget as possible, which is the number of DNNs.

Consistent relative confidence (CRC) [79] is a recently proposed method that only uses output probabilities to rank models. Roughly speaking, given a set of models and input, CRC assumes that a model has higher confidence to input should have better performance. Here, the confidence is calculated by the maximum probability of outputs.

Random sampling is a basic and model-independent method for data selection where each data has an equal probability to be considered. A subset of data is randomly selected and annotated to rank DNNs.

Sample discrimination based selection (SDS) [66] is the state-of-the-art approach in ranking multiple DNNs with respect to accuracy. Following [66], among data in the top 25% with high discrimination scores, we randomly select a given budget of data to label and annotate to perform the ranking task.

Cross Entropy-based Sampling (CES) [80] is designed to select a set of representative data to approximate the actual performance given a single DNN. We follow the same procedure as [66] to adapt CES for multi-DNN comparison.

The comparison we conduct is fair because all methods used the same amount of data information for ranking. All the considered methods need to access the prediction of all unlabeled data once. For sampling-based models, they use this prediction to perform data selection and label a subset of data to rank the models. For labeling-free methods, the ranking is conducted based only on the predictions

and no further labeling budgets are required. Due to the random manner in the sampling methodology, each experiment of the baseline methods is repeated 50 times and we report the average result.

4.3.5 Evaluation Measures

To evaluate the effectiveness of each method, we follow the baseline work [66] and apply the statistical analysis, Spearman's rank-order correlation [81], and Jaccard similarity [66]. The first one evaluates the general ranking of all models, while the last one specifically estimates the ranking of top-k DNNs. In addition, we add the evaluation on Kendall's τ rank correlation [81]. Similar to Spearman's rank-order correlation, Kendall's τ measures the non-parametric rank correlation. However, Kendall's τ calculates based on concordant and discordant pairs and is insensitive to errors (if any) in data. By contrast, Spearman's rank-order correlation calculates based on deviations and is more sensitive to errors (if any) in data.

Given n DNNs, $f_1, f_2, \ldots, f_n$, let $r(f_1), r(f_2), \ldots, r(f_n)$ be the ground truth ranking and
$r'(f_1), r'(f_2), \ldots, r'(f_n)$ be the estimated ranking. The Spearman's rank-order correlation coefficient is computed as

$$\rho = \frac{n\sum_{i=1}^{n} r(f_i)\, r'(f_i) - \left(\sum_{i=1}^{n} r(f_i)\right)\left(\sum_{i=1}^{n} r'(f_i)\right)}{\sqrt{\left[n\sum_{i=1}^{n} r(f_i)^2 - \left(\sum_{i=1}^{n} r(f_i)\right)^2\right]\left[n\sum_{i=1}^{n} r'(f_i)^2 - \left(\sum_{i=1}^{n} r'(f_i)\right)^2\right]}} \tag{4.4}$$

A large ρ indicates that the correlation between the ground truth and estimation is strong.

Kendall's τ is

$$\tau = \frac{P - Q}{\sqrt{(P + Q + T)(P + Q + U)}} \tag{4.5}$$

where P and Q are the numbers of ordered and disordered pairs in $\{r(f_i), r'(f_i)\}$, respectively. T and U are the numbers of ties in $\{r(f_i)\}$ and $\{r'(f_i)\}$, respectively. A large τ indicates a strong agreement between the ground truth and estimation.

Meng *et al.* proposed to apply the Jaccard similarity for measuring the similarity between the top-k models. The similarity coefficient is defined as:

$$J_k = \frac{|\{f_i \mid r(f_i) <= k\} \cap \{f_i \mid r'(f_i) <= k\}|}{|\{f_i \mid r(f_i) <= k\} \cup \{f_i \mid r'(f_i) <= k\}|}, 1 \leq i \leq n \tag{4.6}$$

A large J_k implies a high success in identifying the top-k models.

4.4 Results and Discussion

4.4.1 RQ1: Effectiveness Given ID Test Data

First, we compare the effectiveness of five methods in ranking multiple DNNs based on the accuracy of ID data. Table 4.3 shows the comparison of two labeling-free methods. We can see that for ID test data, LaF significantly outperforms CRC in 6 out of 7 cases with an average 0.565 better correlation score. Only for the Java250 dataset, both LaF and CRC have similar and good results, 0.96 vs. 0.98. Table 4.4 shows Kendall's τ scores of LaF and CRC which also demonstrate LaF is better

Table 4.3: Spearman's correlation coefficient of ranking results of LaF and CRC.

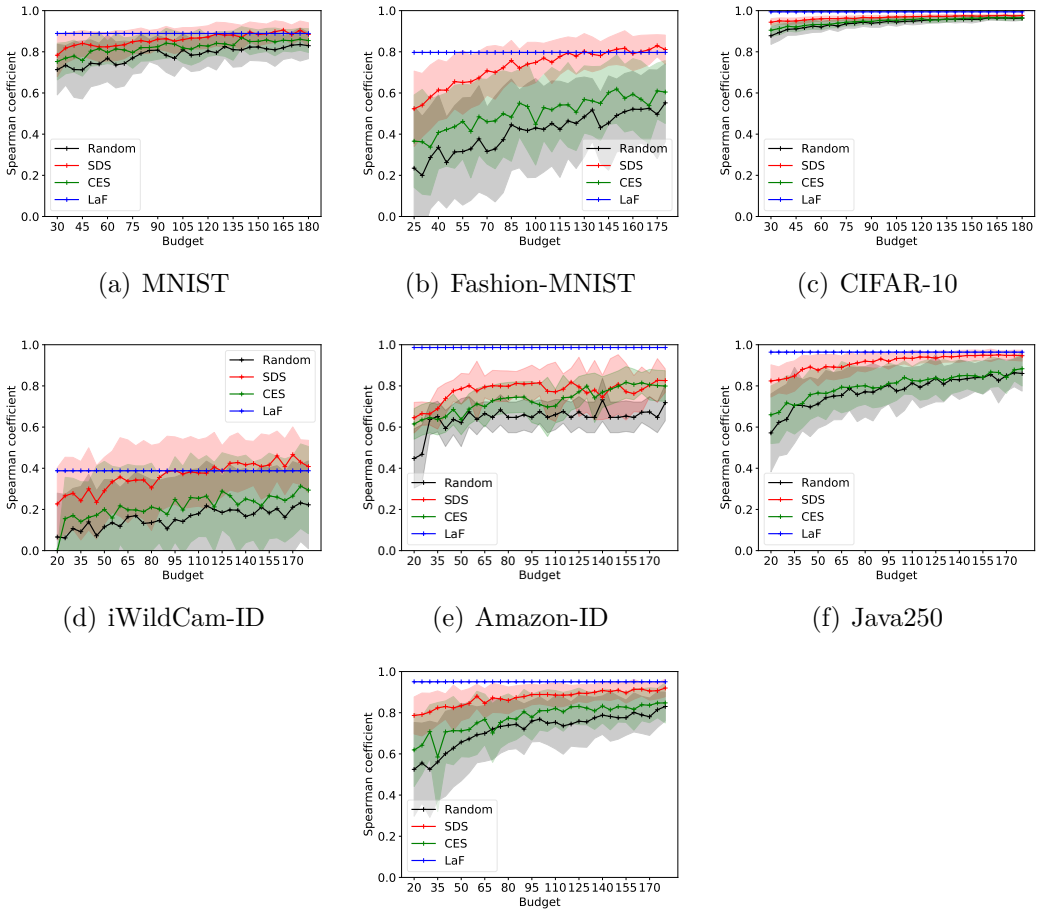
Data Type	**MNIST**		**Fashion-MNIST**		**CIFAR-10**	
	LaF	CRC	LaF	CRC	LaF	CRC
ID	0.89	0.26	0.80	0.36	0.99	0.05
Gaussian Noise	0.97	0.23	0.36	0.23	0.94	0.18
Shot Noise	0.90	0.36	0.33	0.36	0.92	0.18
Impulse Noise	0.97	0.29	0.65	0.29	0.92	0.18
Defocus Blur	0.38	0.16	0.73	0.16	0.99	0.10
Glass Blur	0.77	0.16	0.35	0.16	0.96	0.22
Zoom Blur	0.91	0.37	0.22	0.37	0.99	0.12
Snow	0.98	0.10	0.44	0.10	0.99	0.16
Fog	0.74	0.08	0.34	0.08	0.99	0.05
Brightness	0.98	0.12	0.16	0.12	0.99	0.07
Contrast	0.83	0.13	0.79	0.13	0.95	0.11
Elastic	0.46	0.25	0.71	0.25	0.98	0.16
JPEG	0.90	0.39	0.27	0.39	0.98	0.20
Pixelate	0.92	0.26	0.55	0.26	0.91	0.20
Frost	0.99	0.13	0.35	0.13	0.98	0.22
Motion Blur	0.61	0.20	0.54	0.20	0.98	0.23

	iWildCam		**Amazon**		**Java250**		**C++1000**	
	LaF	CRC	LaF	CRC	LaF	CRC	LaF	CRC
ID	0.39	0.20	0.99	0.39	0.96	0.98	0.95	0.17
OOD	0.91	0.35	0.97	0.36	0.85	0.52	-	-

than CRC. For the comparison of LaF and sampling-based methods, figure 4.3 shows the result measured by Spearman's rank-order correlation. The first conclusion we can draw is that, over seven datasets, all methods succeed in outputting positively correlated rankings. By comparison, LaF continuously outperforms (by up to 0.74) the baseline methods regardless of the labeling budget. Namely, the ranking by LaF is strongly correlated with the ground truth. In general, for the three sample-selection-based baseline methods, the correlation between the estimated rank and the ground truth increases when more data are labeled. However, for some datasets, the performance is still far from LaF. For example, in Amazon, LaF obtains a correlation coefficient of 0.80, while the best baseline, SDS, only achieves 0.48 using the maximum labeling budget of 180. Besides, due to the sampling randomness, each baseline method obtains different ranking results from 50 experiments, which is indicated by the large standard deviation (up to 0.36, shaded area in the figure) at each labeling budget. As a result, the rank by one experiment is not reliable by occasionally being good and poor. In particular, the standard deviation becomes smaller when more data are labeled, which means the ranking method highly relies on the labeling budget. By contrast, since LaF is labeling-free, there is no sampling randomness, in other words, the rank is deterministic.

Additionally, Figure 4.4 presents the effectiveness of all ranking methods based on Kendall's τ rank correlation. By comparison, the result confirms the conclusion drawn from the analysis based on Spearman's rank-order correlation. Namely, our approach stands out concerning the effectiveness without sampling randomness.

Besides, to demonstrate the significance of the two statistical analyses, we cal-

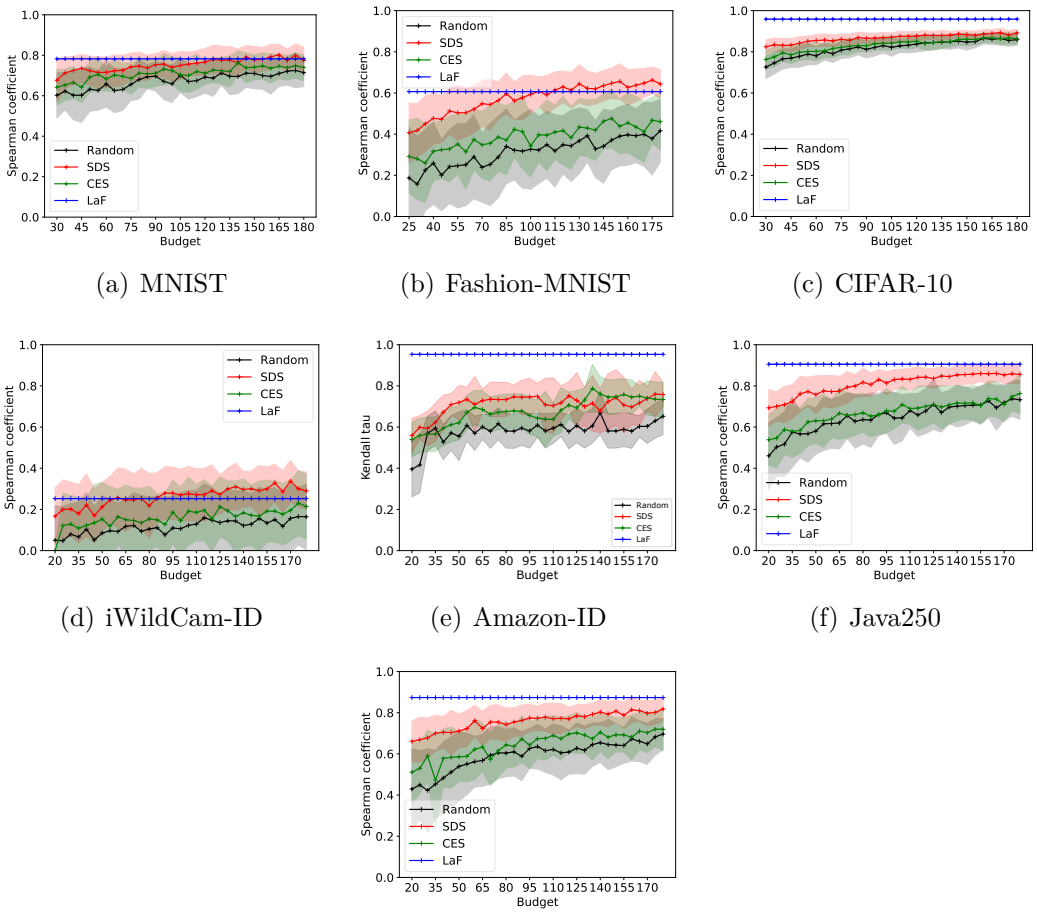


(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

(d) iWildCam-ID (e) Amazon-ID (f) Java250

(g) C++1000

Figure 4.3: Spearman's correlation coefficient of ranking results based on ID test data. The higher the better. The shaded area represents the standard deviation. "Budget" is the number of labeled data.



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

(d) iWildCam-ID (e) Amazon-ID (f) Java250

(g) C++1000

Figure 4.4: Kendall's τ of ranking results based on ID test data. The higher the better. The shaded area represents the standard deviation. "Budget" is the number of labeled data (only apply to Random, SDS, and CES).

Table 4.4: Kendall's τ of ranking results of LaF and CRC.

Data Type	MNIST		Fashion-MNIST		CIFAR-10	
	LaF	CRC	LaF	CRC	LaF	CRC
ID	0.78	0.18	0.61	0.35	0.96	0.03
Gaussian Noise	0.88	0.16	0.38	0.25	0.82	0.13
Shot Noise	0.78	0.25	0.26	0.23	0.84	0.12
Impulse Noise	0.88	0.19	0.58	0.19	0.78	0.13
Defocus Blur	0.35	0.11	0.63	0.25	0.93	0.06
Glass Blur	0.67	0.12	0.38	0.26	0.85	0.15
Zoom Blur	0.82	0.26	0.42	0.27	0.95	0.08
Snow	0.91	0.08	0.36	0.18	0.94	0.12
Fog	0.68	0.06	0.61	0.27	0.94	0.03
Brightness	0.91	0.08	0.18	0.21	0.95	0.07
Contrast	0.76	0.10	0.73	0.17	0.86	0.07
Elastic	0.42	0.17	0.59	0.38	0.93	0.11
JPEG	0.78	0.26	0.26	0.14	0.91	0.13
Pixelate	0.80	0.19	0.40	0.17	0.83	0.13
Frost	0.94	0.09	0.38	0.14	0.91	0.15
Motion Blur	0.51	0.14	0.52	0.18	0.92	0.15

	iWildCam		Amazon		Java250		C++1000	
	LaF	CRC	LaF	CRC	LaF	CRC	LaF	CRC
ID	0.25	0.19	0.95	0.29	0.91	0.92	0.87	0.17
OOD	0.77	0.27	0.92	0.28	0.83	0.55	-	-

culate the corresponding p-value of all methods. A p-value lower than the common significance level of 0.05 indicates that the ranking is strongly correlated with the ground truth. Except for the iWildCam dataset, the ranking results by LaF are all strongly correlated. However, due to the effectiveness and sampling randomness, the baseline methods always achieve insignificant rankings. For the iWildCam dataset, we believe the reason is that the difference between multiple DNNs is too slight given 182 classes. For instance, the accuracy difference between the best and worst is only 1.54% (Table 4.1). The impact of the accuracy/robustness on the ranking is investigated in Section 4.4.4.

On the other hand, we evaluate different methods concerning identifying the top-k DNNs ($k = 1, 3, 5, 10$). Table 4.5 lists the result of Jaccard similarity. On average, LaF achieves the best result regardless of the datasets. It is better than the worst performance by up to 0.33, 0.32, 0.33, and 0.27 in the top 1, 3, 5, and 10 rankings, respectively. Concretely, in the top-1 ranking, for datasets MNIST, Fashion-MNIST, and iWildCam, all methods (Random, SDS, and ours) are not effective (under 0.08). Remark that CES takes the best results of all models for each labeling budget when knowing the ground truth. Specifically, it takes n (number of DNNs) times of labeling budget. Therefore, it sometimes outperforms others but is not applicable in practice.

Answer to RQ1: Based on the accuracy of ID test data, LaF outperforms all baseline methods in outputting strongly correlated ranking. In addition, statistical analysis demonstrates that outperforming is significant.

Table 4.5: Jaccard similarity of ranking the top-k DNNs based on the clean accuracy. For baseline methods, we report the average results over all labeling budgets. The best performance is highlighted in gray. The higher the better.

Jaccard	Method	MNIST	Fashion-MNIST	CIFAR-10	iWildCam	Amazon	Java250	C++1000	Average
k=1	Random	0.01	0.02	0.19	0.07	0.12	0.12	0.09	0.10
	SDS	0.03	0.07	0.17	0.02	0.20	0.18	0.20	0.15
	CES	0.65	0.23	0.21	0.10	0.15	0.84	0.94	0.45
	Our	0.00	0.00	1.00	0.00	1.00	0.00	1.00	0.43
k=3	Random	0.07	0.12	0.36	0.14	0.17	0.21	0.19	0.19
	SDS	0.11	0.24	0.37	0.20	0.31	0.88	0.29	0.27
	CES	0.69	0.28	0.39	0.15	0.21	0.18	0.80	0.46
	Our	0.20	0.20	1.00	0.20	1.00	0.70	0.50	0.51
k=5	Random	0.11	0.19	0.50	0.22	0.25	0.23	0.30	0.28
	SDS	0.17	0.33	0.60	0.33	0.39	0.79	0.44	0.39
	CES	0.89	0.36	0.55	0.16	0.29	0.18	0.68	0.53
	Our	0.25	0.43	1.00	0.25	1.00	0.70	0.67	0.61
k=10	Random	0.23	0.35	0.80	0.43	0.43	0.26	0.58	0.49
	SDS	0.41	0.55	0.85	0.46	0.49	0.65	0.79	0.62
	CES	0.73	0.46	0.82	0.44	0.46	0.24	0.68	0.61
	Our	0.67	0.54	1.00	0.43	0.67	1.00	1.00	0.76

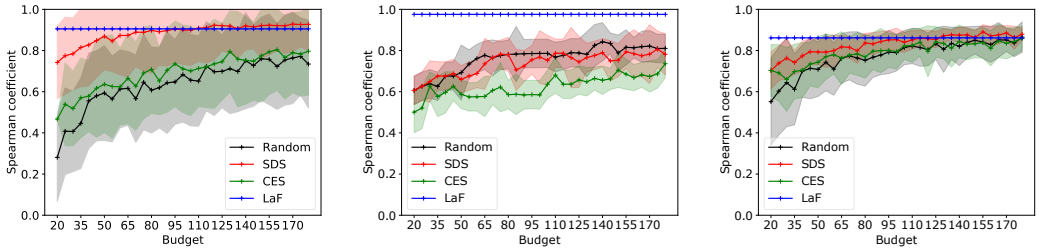
4.4.2 RQ2: Effectiveness Under Distribution Shift

For the synthetic distribution shift, table 4.3 and table 4.4 show the results of two labeling-free methods. The results indicate that LaF outperforms CRC in 46 out of 49 cases in terms of Spearman's correlation coefficient, and 48 out of 49 cases in terms of Kendall's results. Considering the comparison of LaF and sampling-based methods, tables 4.6 and 4.7 summarize the results of Spearman's rank-order correlation and Kendall's τ on MNIST-C, Fashion-MNIST-C, and CIFAR-10-C, respectively. We observe that our approach achieves the best performance in most cases, for instance, in CIFAR-10-C, LaF consistently beats other methods. Furthermore, as shown in RQ1, SDS performs the second best among the four ranking approaches. However, compared to random and CES, SDS tends to lose its performance in these two tables (highlighted in yellow). For example, in MNIST-C with Defocus Blur, SDS ranks the models wrongly with a correlation of -0.15 (Table 4.6), while CES can achieve a good ranking performance. The reason is that SDS only relies on the prediction of models to rank. However, the confidence of models on OOD data reduces when the level of severity increases, e.g., SDS performs extremely poorly on MNIST-C where many models have very low accuracy on the dataset MNIST-C-severity-5. In short, this existing state-of-the-art approach is sensitive to artificial distribution shifts, which calls for the testing under distribution shifts of existing approaches. Considering the Jaccard similarity, in the 75 corruptions of MNIST-C, both LaF and CES outperform the random sampling and SDS to identify the top DNNs precisely. In CIFAR-10-C, LaF achieves the best performance (similarity of 1) in most cases (173 of 300). For the natural distribution shift, the results are shown in Figure 4.5. LaF can better distinguish the performance of DNNs than the baseline methods. In addition, concerning the Jaccard similarity in Table 4.8, LaF is also the best in identifying the top DNNs. The reason for the good performance of LaF is that the Bayesian model in LaF can model the probability of each possible label given the predicted labels from different DNNs and model parameters (data difficulty and model specialty in LaF) which makes LaF flexible to the change of predictions.

In addition, compared to the effectiveness given ID test data, the ranking by all methods is different since the performance of DNNs changes given OOD test data. However, we notice an opposite phenomenon happens. Given ID test data,

Table 4.6: Spearman's correlation coefficient of ranking results based on MNIST-C, Fashion-MNIST-C, and CIFAR-10-C. For baseline methods, we compute the average and standard deviation over all labeling budgets and 50-repetition experiments. The best performance is highlighted in gray. Values highlighted in yellow indicate CES or random outperform SDS. The higher the better.

	MNIST				**Fashion-MNIST**				**CIFAR10**			
	Random	SDS	CES	LaF	Random	SDS	CES	LaF	Random	SDS	CES	LaF
Gaussian Noise	0.90	0.94	0.91	0.97	0.29	0.26	0.39	0.36	0.91	0.93	0.92	0.94
Shot Noise	0.76	0.83	0.83	0.90	0.34	0.29	0.42	0.33	0.90	0.92	0.91	0.95
Impulse Noise	0.38	0.44	0.94	0.97	0.26	0.27	0.33	0.65	0.89	0.90	0.90	0.92
Defocus Blur	0.84	0.10	0.87	0.38	0.25	0.30	0.31	0.73	0.93	0.95	0.94	0.99
Glass Blur	-0.16	-0.15	0.93	0.77	0.31	0.31	0.36	0.35	0.84	0.93	0.87	0.96
Zoom Blur	0.77	0.83	0.88	0.91	0.27	0.24	0.39	0.22	0.92	0.94	0.93	0.99
Snow	0.50	0.51	0.95	0.98	0.25	0.23	0.32	0.44	0.91	0.92	0.92	0.99
Fog	0.40	0.39	0.98	0.74	0.16	0.23	0.26	0.34	0.95	0.97	0.96	0.99
Brightness	0.94	0.47	0.95	0.98	0.04	0.05	0.13	0.16	0.94	0.97	0.95	0.99
Contrast	0.94	0.35	0.95	0.83	0.28	0.33	0.26	0.79	0.93	0.95	0.94	0.95
Elastic	0.80	0.16	0.83	0.46	0.41	0.39	0.29	0.71	0.90	0.94	0.92	0.98
JPEG	0.78	0.85	0.82	0.90	0.36	0.34	0.39	0.27	0.79	0.89	0.82	0.98
Pixelate	0.38	0.40	0.86	0.92	0.32	0.27	0.32	0.55	0.89	0.90	0.91	0.91
Frost	0.44	0.42	0.98	0.99	0.23	0.24	0.29	0.35	0.90	0.92	0.92	0.98
Motion Blur	0.58	0.62	0.95	0.61	0.27	0.32	0.32	0.54	0.92	0.94	0.93	0.98
Average	0.62	0.48	0.91	0.82	0.27	0.27	0.32	0.45	0.90	0.93	0.92	0.97



(a) iWildCam-OOD (b) Amazon-OOD (c) Java250-OOD

Figure 4.5: Spearman's correlation coefficient of ranking results based on OOD test data. The higher the better. The shaded area represents the standard deviation. "Budget" is the number of labeled data.

LaF achieves 0.39, 0.99, and 0.96 concerning Spearman's coefficient for iWildCam, Amazon, and Java250, respectively. When given OOD test data, the results are 0.91, 0.97, and 0.85, respectively. In other words, the effectiveness improves on the OOD test data in iWildCam but degrades in Amazon and Java250. To make clear the reason behind this, we analyze the accuracy and robustness of multiple DNNs on ID and OOD test data (Table 4.1), respectively. In iWildCam, the performance difference of its 20 DNNs becomes larger on OOD test data, from 1.54% to 11.52%. In Amazon, the performance of all 20 DNNs degrades, e.g., from 56.06% to 59.13%. Besides, the performance difference in Amazon becomes smaller. Therefore, we believe that the model's ability and the performance difference among DNNs have an impact on the ranking effectiveness, which leads to the investigation in RQ3.

Table 4.7: Kendall's τ of ranking results based on MNIST-C, Fashion-MNIST-C, and CIFAR-10-C. For Random, SD, and CES, we compute the average and standard deviation over all labeling budgets and 50-time experiments. The best performance is highlighted in gray. Values highlighted in yellow indicate CES or random outperform SDS. The higher the better.

	MNIST				**Fashion-MNIST**				**CIFAR10**			
	Random	SDS	CES	LaF	Random	SDS	CES	LaF	Random	SDS	CES	LaF
Gaussian Noise	0.79	0.84	0.82	0.88	0.27	0.27	0.33	0.38	0.78	0.79	0.80	0.82
Shot Noise	0.64	0.71	0.71	0.78	0.34	0.30	0.37	0.26	0.76	0.77	0.78	0.84
Impulse Noise	0.32	0.36	0.84	0.88	0.28	0.28	0.33	0.58	0.75	0.74	0.77	0.78
Defocus Blur	0.71	0.10	0.74	0.35	0.27	0.30	0.31	0.63	0.81	0.85	0.82	0.93
Glass Blur	-0.11	-0.10	0.82	0.67	0.31	0.30	0.34	0.38	0.70	0.81	0.73	0.85
Zoom Blur	0.65	0.71	0.77	0.82	0.29	0.25	0.36	0.42	0.80	0.81	0.81	0.95
Snow	0.43	0.44	0.86	0.91	0.27	0.25	0.33	0.36	0.77	0.79	0.79	0.94
Fog	0.32	0.31	0.91	0.68	0.27	0.24	0.28	0.61	0.84	0.88	0.85	0.94
Brightness	0.86	0.41	0.87	0.91	0.07	0.07	0.13	0.18	0.82	0.87	0.83	0.95
Contrast	0.85	0.31	0.86	0.76	0.30	0.33	0.29	0.73	0.81	0.85	0.83	0.86
Elastic	0.70	0.15	0.72	0.42	0.40	0.38	0.31	0.59	0.76	0.81	0.78	0.93
JPEG	0.66	0.74	0.70	0.78	0.35	0.34	0.35	0.26	0.63	0.73	0.66	0.91
Pixelate	0.33	0.35	0.74	0.80	0.35	0.28	0.38	0.40	0.76	0.76	0.78	0.83
Frost	0.37	0.36	0.91	0.94	0.25	0.27	0.31	0.38	0.77	0.79	0.79	0.91
Motion Blur	0.48	0.51	0.83	0.51	0.29	0.33	0.33	0.52	0.79	0.82	0.81	0.92
Average	0.53	0.41	0.81	0.74	0.29	0.28	0.32	0.45	0.77	0.81	0.79	0.89

Answer to RQ2: Under different distribution shifts, LaF still outstands in all ranking methods. Due to the distribution shift, DNNs' performance declines, which degrades the ranking effectiveness. Particularly, SDS is sensitive to artificial distribution shifts and fails to defeat random sampling and CES in many cases.

4.4.3 RQ3: Ablation Study

To check if each component contributes to the performance of LaF, we conduct an ablation study. Specifically, we prepare two variants of LaF,

- LaF without pruning. We remove the pruning process of LaF which means LaF will use all the inputs to do majority voting.
- LaF without optimizing. We remove the optimizing process of LaF and directly use the results of the majority voting to perform the final performance ranking.

Table 4.9 and Table 4.10 summarize the results of our ablation study on ID test data. We can see in most cases (12 out of 14 cases), LaF outperforms the other two variants. This means each component of LaF is necessary and has a positive impact on the performance of LaF.

Answer to RQ3: Each component contributes to the performance of LaF and can not be removed.

4.4.4 RQ4: Analysis of Impact factors

In this part, we analyze the potential factors that could affect the effectiveness of LaF, 1) the number of candidate models, 2) the number of input data used for ranking, 3) the quality of candidate models, and 4) the diversity of candidate models.

For the analysis of the first two factors, we reduce the number of candidate models and test inputs and then repeat the model ranking experiments. Concretely, we set the ratio of models and inputs as 20%, 40%, 60%, 80%, and 100%, and use LaF to rank the models and compare the results to check if these two factors have a high impact on the performance of LaF. Note that, 100% of inputs and models mean

Table 4.8: Jaccard similarity of ranking the top-k DNNs concerning natural distribution shift. For baseline methods, we report the average results over all labeling budgets. The best performance is highlighted in gray. The higher the better.

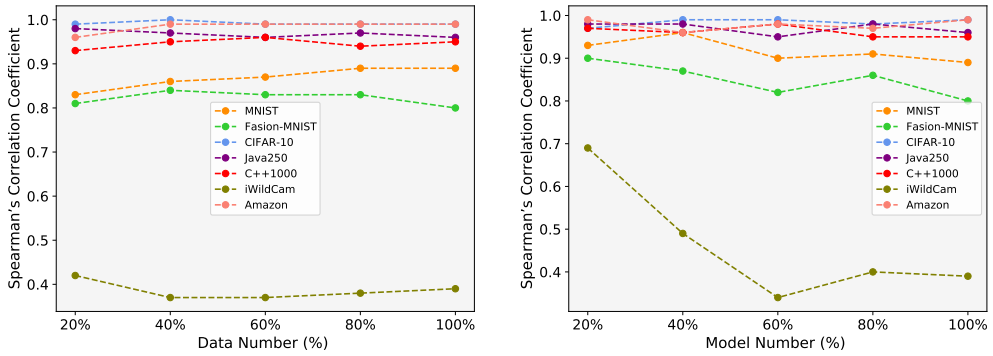
Jaccard	**Dataset**	**Random**	**SDS**	**CES**	**LaF**
k=1	iWildCam	0.66	0.96	0.68	1
	Amazon	0.01	0.03	0.84	0.00
	Java250	0.16	0.00	0.95	0.00
k=3	iWildCam	0.4	0.63	0.41	1
	Amazon	0.14	0.21	0.86	0.67
	Java250	0.27	0.11	0.76	0.00
k=5	iWildCam	0.44	0.61	0.47	0.67
	Amazon	0.22	0.19	0.31	0.67
	Java250	0.36	0.28	0.78	0.25
k=10	iWildCam	0.62	0.83	0.65	0.82
	Amazon	0.41	0.12	0.51	0.81
	Java250	0.61	0.72	0.67	1

Table 4.9: Results of ablation study. Spearman's correlation coefficient of ranking results. The best performance is highlighted in gray.

	LaF	w/o Pruning	w/o Optimizing
MNIST	0.89	0.88	0.85
Fashion-MNIST	0.80	0.80	0.77
CIFAR-10	0.99	0.98	0.99
Java250	0.96	0.97	0.94
C++1000	0.95	0.90	0.94
iWildCam	0.39	0.34	0.35
Amazon	0.99	0.95	0.92

Table 4.10: Results of ablation study. Kendall's τ of ranking results. The best performance is highlighted in gray.

	LaF	w/o Pruning	w/o Optimizing
MNIST	0.78	0.72	0.77
Fashion-MNIST	0.96	0.96	0.95
CIFAR-10	0.61	0.60	0.61
Java250	0.91	0.86	0.89
C++1000	0.87	0.89	0.84
iWildCam	0.25	0.25	0.18
Amazon	0.95	0.93	0.91



(a) Impact of data number

(b) Impact of model number

Figure 4.6: Impact of numbers of data and models. Spearman's correlation coefficient of ranking results.

the settings used in the first three research questions. Figure 4.6 depicts the results of ID test data. First, considering the input numbers, we can see the results are relatively stable. With increasing (or decreasing) the number of inputs, the ranking performance of LaF is still at a similar level. This indicates LaF is not sensitive to the number of input data. On the other hand, considering the number of models, the results indicate that LaF is more effective to rank a small number of models. With the increase of model numbers, the performance of LaF slightly drops, except for the iWildCam which has a big performance drop. We can conclude that it is more challenging to rank a large number of models by using LaF.

As mentioned in RQ2, by comparing the ranking effectiveness given ID and OOD test data, we raise the demand of investigating the two impact factors, the quality, and diversity of multiple DNNs. The quality refers to the model's performance given the test data and is calculated as the average accuracy or robustness over all DNNs on each dataset. For instance, in MNIST without distribution shift, the quality is the average accuracy of 30 DNNs on the ID test data, and in MNIST-C with Gaussian Noise (severity=1), the quality is the average robustness of 30 DNNs on the corresponding OOD data. The diversity indicates the performance difference among DNNs and is the standard deviation of accuracy or robustness over all DNNs on each dataset.

Figure 4.7 plots the distribution of ranking performance concerning quality and diversity. Most good rankings happen with a high model quality (greater than 50%). The reason is that in our scenario, we only have access to the predictions of test data of multiple DNNs, which setups the initial inference of data difficulty and model specialty. Therefore, the learned Bayesian model can be more precise when the qualities of DNNs are high. Furthermore, this also explains why LaF outperforms the sampling-based methods. For example, SDS selects a few discriminative data to annotate to rank DNNs and the selection of data highly relies on the predicted labels. As a result, since the low qualities of DNNs always give a wrong estimation of the discrimination ability of data, the ranking performance is poorer. For instance, in Java250 and C++1000, SDS only reaches 0.82 and 0.79 on Spearman's correlation with 20 labeled data, respectively. However, LaF achieves 0.96 and 0.95 in two datasets, respectively, with no labeling effort. On the other hand, concerning the diversity, Figure 4.7 reveals that there is a high chance of a good ranking when DNNs

are diverse (greater than 5%). Additionally, a poor ranking mostly happens when DNNs are too close to each other, which confirms the result of iWildCam with ID data (Figure 2.3(d)) that all ranking methods obtain poor ranking.

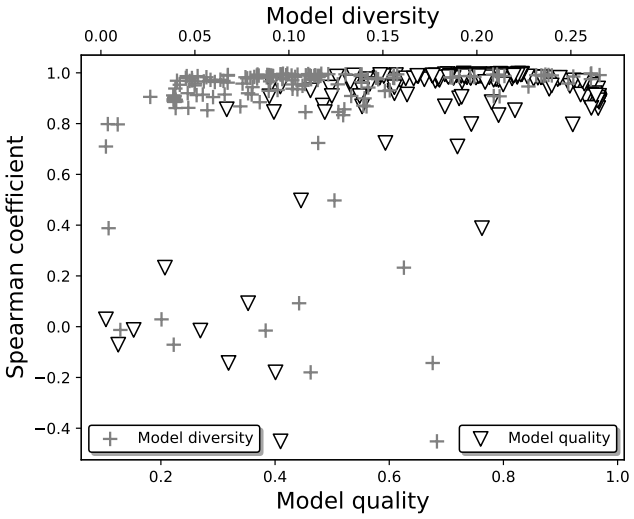


Figure 4.7: The impact of model quality/diversity on the ranking performance of LaF. Each point indicates Spearman's correlation coefficient of a specific dataset and its DNNs. All 160 datasets are included.

Answer to RQ4: There is no connection between the performance of LaF and the number of used test data. On the other hand, with the increase in model numbers, LaF is more challenging to rank the models. Besides, when the multiple DNNs have high quality (e.g., the average accuracy/ robustness is over 50%), the performance of DNNs can be discriminated better. On the other hand, there is a higher chance of a good ranking when DNNs are more diverse (larger than 18%).

4.4.5 Applicable Scenarios

Collecting a large number of labeled data is the starting point to build a high-performance DNN model. However, data labeling is time-consuming and usually requires domain knowledge. We provide some application scenarios of our work.

1. Overcome the challenge of insufficient domain knowledge. For example, a Java developer can easily annotate the source code in Java but will have difficulties with other programming languages such as C++, due to the difference in handling memory allocations. Consider a scenario where a Java developer builds a DNN model to solve some Java tasks, e.g., bug detection. If the developer later switches from Java to C++, preparing labeled C++ datasets to select another good model from the open source can be challenging. LaF can assist in this scenario by enabling the developer to select a suitable model without this domain knowledge.

2. Manage distribution shift. Consider a company that intends to leverage a DNN model to solve a particular task. They prepared some labeled data to select and build a good model for this month. However, as time goes by, the new coming data may follow a different distribution from the existing labeled data.

If the company wishes to switch to a better model, it needs to label additional new data and redo the model selection. With LaF, the company can effortlessly obtain the best model without further labeling effort.

4.5 Discussion

4.5.1 Limitation

Since LaF relies on the results of majority voting to annotate the inputs, the quality of voting could affect the performance of LaF. When many candidate models (not all the models since LaF has a pruning process to filter these data where all the models have the same prediction) have consistently wrong predictions on a large number of inputs, LaF could have a poor ranking performance. However, this is the corner case that we will rarely meet. Even though models have poor performance, their wrong predictions can be different, for example, in MNIST-C-Defocus Blur-Severity=5, almost all of the models have bad performance (accuracy ranges from 1.96-18.30), LaF still performs well for ranking them.

4.5.2 Threats to validity

The internal threat is mainly due to the implementation of the baseline methods, our proposed approach, and the evaluation metrics. For SDS, we use the original implementation on GitHub provided by Meng *et al.* [66]. For random sampling and CES, we implement it based on the description in [66] and carefully check the result to be consistent with that in [66]. Regarding the evaluation metrics, we adopt popular libraries, such as SciPy [82].

The external threat comes from the evaluated tasks, datasets, DNNs, and baseline methods. Regarding the classification tasks, we consider three different ones, image, text, and source code. For the datasets, we select the publicly available datasets. In particular, for datasets with the artificial distribution shift (15 types of natural corruptions) and natural distribution shift, we employ four public benchmarks. Concerning the DNNs, we collect them (either the off-the-shelf models or train with the provided scripts) from different repositories on GitHub. These models are with different architectures and parameters. For the comparison, we consider three sample-selection-based baseline methods and apply different numbers of labeling budgets to imply their performance.

The construct threat mainly lies in the sampling randomness in the baseline methods and the evaluation measures. To reduce the impact of randomness, for each baseline method, we repeat each experiment concerning the labeling budgets, and datasets 50 times and report the results of both average and standard deviation. Since our proposed approach does not rely on sampling data to annotate, there is no sampling randomness. Considering the randomness (gradient ascent search) in the EM algorithm, we repeat LaF 50 times and found that the randomness was negligible (less than 1.84E-03). Regarding the evaluation measures, we consider three popular statistical analyses. Kendall's τ rank correlation and Spearman's rank-order correlation can infer the effectiveness of the methods concerning the general ranking, while the Jaccard similarity can specifically check the performance concerning the top-k ranking. Besides, for the statistical analyses, we report the p-value to demonstrate the significance.

4.6 Conclusion and Future Work

Observing the limitations (labeling effort, sampling randomness, and performance degradation on out-of-distribution data) of existing selection-based methods, we proposed a labeling-free approach to undertake the task of ranking multiple deep neural networks (DNNs) without the need for domain expertise to lighten the MLOps. The main idea is to build a Bayesian model given the predicted labels of data, which allows for free labeling and non-sampling randomness. The experimental results on various domains (image, text, and source code) and different performance criteria (accuracy and robustness against artificial and natural distribution shifts) demonstrate that LaF significantly outperforms the three baseline methods concerning both Spearman's correlation and Kendall's τ. In addition, the results of the Jaccard similarity show the efficiency of LaF in identifying the top-k ($k = 1, 3, 5, 10$) DNNs. This work currently only focuses on the classification task, we will explore it for other tasks, such as regression, in future work. Observing the ranking difference on ID and out-of-distribution (OOD) test data, our approach might be useful to detect the existence of distribution shifts. We will consider this in future work.

In the future, we plan to:

- Combine sampling-based methods and LaF to further improve the effectiveness of model ranking. Since we can see that in some cases (e.g., models have low quality and diversity) LaF cannot perform well, we can use β to check the quality of models first, and then use sampling-based methods to rank models for these cases.
- Utilize LaF to build better model ensembles. Since existing works [83] show that it is good to use diverse models to build model ensembles, we can leverage LaF (β value) to check the model diversity for boosting ensemble building.

4.7 Appendix

4.7.1 Increasing Labeling Budgets

In this section, we provide more results by comparing LaF to sampling-based methods with bigger labeling budgets. Here, we consider labeling budgets 500, 1500, and 2500. Note that SDS needs to select data from the top 25% ranked data, thus labeling budget of 2500 is not suitable for the iWildCam dataset (over 2500 data). Table 4.11 summarizes the results. We can see that in most cases (14 out of 21 cases), LaF outperforms sampling-based methods under labeling budget 500. Under the labeling budget of 1500, LaF still has competitive performance with other baselines, e.g., LaF achieves the best results in 10 out of 21 cases. However, when the labeling budget becomes 2500, sampling-based methods are better choices.

4.7.2 Accuracy of Artificial Distribution Shift Datasets

Table 4.12 summarizes the accuracy of our collected models on artificial distribution shift datasets.

Table 4.11: Increasing the budgets of sampling-based methods. Values highlighted by yellow backgrounds indicate where LaF is better.

Dataset	**Method**	**500**	**1500**	**2500**	**Dataset**	**Method**	**500**	**1500**	**2500**
MNIST	Random	0.89	0.95	0.98	**C++1000**	Random	0.85	0.91	0.96
	CES	0.96	0.97	1.00		CES	0.95	0.95	0.96
	SDS	0.95	0.97	1.00		SDS	0.96	0.98	0.98
	LaF	0.89	0.89	0.89		LaF	0.95	0.95	0.95
CIFAR-10	Random	0.98	0.99	1.00	**iWildCam**	Random	0.36	0.53	0.58
	CES	0.99	0.99	1.00		CES	0.54	0.60	0.60
	SDS	0.99	0.99	0.99		SDS	0.46	0.45	-
	LaF	0.99	0.99	0.99		LaF	0.39	0.39	0.39
Fashion-MNIST	Random	0.77	0.79	0.93	**Amazon**	Random	0.56	0.76	0.85
	CES	0.87	0.86	0.94		CES	0.75	0.85	0.93
	SDS	0.90	0.91	0.91		SDS	0.56	0.79	0.86
	LaF	0.80	0.80	0.80		LaF	0.99	0.99	0.99
Java250	Random	0.91	0.96	0.97					
	CES	0.96	0.97	0.99					
	SDS	0.95	0.96	0.96					
	LaF	0.96	0.96	0.96					

Table 4.12: Summary of MNIST-C, Fashion-MNIST-C, and CIFAR-10-C with the artificial distribution shift and robustness. Each dataset includes 15 types of natural corruptions (e.g., Gaussian Noise) with 5 levels of severity (1-5). The number in each cell presents the minimum and maximum robustness of multiple DNNs given the corruption type and severity.

Corruption Type	Severity=1	Severity=2	Severity=3	Severity=4	Severity=5
MNIST-C					
Gaussian Noise	84.85-99.49	80.49-99.25	55.89-99.05	29.86-98.77	16.92-96.02
Shot Noise	84.89-99.53	84.73-99.49	84.87-99.38	84.32-99.00	83.12-98.68
Impulse Noise	84.29-99.20	71.33-98.91	56.50-98.62	27.77-96.39	16.80-88.67
Defocus Blur	58.63-95.69	31.76-84.91	9.73-43.40	3.73-20.46	1.96-18.30
Frosted Glass Blur	65.06-96.52	54.02-94.33	19.06-75.43	15.63-70.29	11.12-54.46
Motion Blur	56.71-97.29	36.28-90.04	25.23-78.19	20.29-65.58	18.59-61.07
Zoom Blur	82.68-99.54	81.50-99.45	80.42-99.35	78.30-99.22	74.93-98.7
Snow	51.43-99.06	17.78-98.38	19.27-96.03	16.53-93.73	11.39-95.73
Frost	17.25-98.99	10.60-97.91	9.52-96.72	9.18-96.63	9.05-95.36
Fog	9.74-97.07	9.61-95.32	9.75-89.51	9.73-85.28	9.69-73.37
Brightness	66.18-99.53	21.94-99.22	12.55-98.99	9.96-98.60	9.10-97.06
Contrast	37.47-99.22	20.38-99.17	10.29-99.04	9.74-98.08	8.02-92.11
Elastic	49.09-99.07	12.80-17.58	79.89-97.75	44.52-94.33	12.32-71.43
Pixelate	80.40-98.64	84.37-99.11	77.65-98.47	62.11-92.34	54.96-89.33
JPEG	84.90-99.53	84.93-99.56	85.10-99.54	84.78-99.44	84.85-99.36
Fashion-MNIST-C					
Gaussian Noise	74.57-74.57	56.19-83.34	37.41-73.87	18.24-61.61	11.94-44.15
Shot Noise	82.48-89.97	77.72-87.53	72.08-83.93	62.20-78.25	53.33-72.39
Impulse Noise	52.35-91.07	29.11-87.62	20.06-83.07	12.87-69.44	10.81-54.32
Defocus Blur	26.08-70.49	13.51-59.18	7.39-44.41	3.35-33.09	1.87-27.16
Frosted Glass Blur	59.16-80.66	57.87-77.73	28.12-58.08	28.37-58.97	23.0-52.96
Motion Blur	70.43-92.55	66.97-91.59	61.61-90.74	58.67-89.68	53.02-88.42
Zoom Blur	61.09-87.87	34.25-77.51	41.80-78.9	40.72-78.17	34.46-77.66
Snow	29.40-82.47	23.91-78.08	19.23-69.08	17.66-59.28	14.38-43.6
Frost	64.64-90.34	54.74-89.14	40.69-88.19	31.25-87.18	27.16-85.87
Fog	29.86-87.42	21.40-85.32	12.80-80.58	10.00-62.81	10.00-42.9
Brightness	51.79-85.74	15.64-19.97	51.55-77.36	41.33-67.57	22.46-50.32
Contrast	83.89-90.14	81.87-89.8	80.78-89.16	76.59-88.1	74.80-87.01
Elastic	40.71-64.36	76.53-88.27	70.78-84.76	27.80-51.18	40.37-72.85
Pixelate	47.18-88.03	33.51-84.25	27.71-79.91	27.11-79.0	23.45-75.86
JPEG	44.56-84.03	31.41-72.04	29.23-58.28	22.68-47.94	18.36-45.64
CIFAR-10-C					
Gaussian Noise	59.02-82.46	45.70-75.57	34.70-73.70	30.07-72.54	25.08-71.14
Shot Noise	66.72-88.90	57.29-81.71	42.12-75.33	37.34-74.12	31.77-71.97
Impulse Noise	63.82-88.70	52.58-83.19	42.74-76.43	24.28-66.53	15.32-61.54
Defocus Blur	69.62-96.13	68.32-96.19	66.24-95.85	52.11-91.50	31.98-86.14
Frosted Glass Blur	35.53-74.17	37.23-73.88	40.17-73.61	30.33-68.81	32.24-68.36
Motion Blur	67.30-94.31	59.71-91.27	49.23-86.62	48.20-86.27	40.05-81.53
Zoom Blur	64.45-93.94	57.73-94.29	49.07-92.93	42.70-91.18	35.24-87.35
Snow	67.65-93.63	54.93-87.77	59.97-88.74	58.91-86.40	54.09-82.56
Frost	66.73-92.93	62.62-89.90	54.64-84.75	53.15-83.61	44.76-76.56
Fog	68.71-95.87	61.93-95.16	55.08-93.99	45.97-91.18	32.22-75.58
Brightness	69.55-95.76	68.88-95.65	67.51-94.98	65.73-94.44	60.59-92.64
Contrast	67.12-95.77	49.64-93.63	38.69-91.56	30.37-87.35	15.53-64.79
Elastic	62.71-93.92	63.91-94.03	63.78-93.46	55.80-86.46	53.64-80.07
Pixelate	69.62-94.77	67.38-91.97	60.46-89.24	42.55-78.97	28.45-75.55
JPEG	69.29-87.71	64.43-83.60	61.23-81.53	59.08-80.11	54.93-77.30

5 Towards Exploring the Limitations of Active Learning: An Empirical Study

In this chapter, we conduct a comprehensive empirical study to explore the limitations of active learning. Specifically, 1) we scrutinize how active learning influences the adversarial robustness of deep neural network (DNN) models. 2) We assess whether models trained using active learning can uphold their accuracy levels even after undergoing compression. 3) We check the relationship between the quantity of training data, the resilience of models against adversarial attacks, and their sustained accuracy post-compression.

Contents

5.1 Introduction

Deep learning (DL) has achieved tremendous success in various cutting-edge application domains, such as image processing [1], machine translation [84], autonomous vehicles [18], and robotics [85]. Two key elements to achieve high-performing predictions are well-designed deep neural networks (DNNs) (with appropriate architecture and parameters) and a carefully chosen set of labeled training data. However, data labeling is expensive and time-consuming because it requires a large amount of human effort. For example, it took more than 3 years to prepare the first version of the ImageNet [86] dataset. Thus, acquiring labeled data is seen as a major obstacle to the widespread adoption of DL [87].

One established solution to reduce data labeling cost is active learning [88], i.e., incremental methods to select informative subsets of training data to undergo labeling in a way that the produced model is as accurate as if it was trained on all data. With active learning, engineers can thus compromise labeling effort with model performance (e.g., classification accuracy). There has been much research on devising active learning methods, each relying on different data selection metrics. These metrics [89, 90, 91, 92] typically exploit information of the DNN under training (e.g., its gradient or uncertainty) to select the most informative data to label next.

On the other hand, recent work on DL testing and debugging [21, 93, 94, 17, 16, 95, 96, 97] have proposed different metrics for *test generation* and *test selection*, i.e., the problem of selecting test data that are more likely to be misclassified by the model [98]. As in active learning scenarios, these test data can then be used to improve the model (by retraining).

The proliferation of data selection metrics (coming from active learning and testing) makes it challenging for engineers to decide which one they should use. Indeed, the different metrics have been evaluated on a restricted set of problems (mostly image classification datasets) under incomparable experimental settings (different models and labeling budget) [96, 97]. There is, therefore, a need for a comprehensive study of all these data selection metrics on a common ground involving different classification tasks.

Another gap in the current body of knowledge is that most experimental studies evaluate the metric wrt. the test accuracy of the trained models.[1] Other key quality indicators have been ignored, such as the model robustness to adversarial attacks and the model performance after compression. The lack of consideration for these indicators raises practical issues when deploying DL models trained using active learning (consider, e.g., biomedical image segmentation [100] or mobile DL applications [101]). Hence, one should make sure that active learning can produce models whose quality is not limited to classification accuracy but extends to *all* quality indicators relevant to the use case.

To fill these gaps, in this paper, we conduct a comparative empirical study to explore the potential limitations of active learning. Our study involves 15 data selection metrics evaluated over 5 datasets (2 image classification tasks and 3 text classification tasks), 10 DNN architectures, and 20 labeling budgets (ratio of training data that can be labeled). Specifically, our study aims to answer the following four research questions:

• **RQ1: How effective are the different data selection metrics for producing**

[1] Rarely, some studies [99] measure empirically the robustness of the model to adversarial attacks.

accurate models? We answer this question by measuring how fast (i.e., how many training data are required) the accuracy of the trained model converges to the accuracy of the fully trained model (i.e., trained with the full training set). Besides, we employ random selection as a baseline to compare the effectiveness of each metric. Our results indicate significant differences between metrics (up to 45.9% of test accuracy), especially when less than 50% of the training data is used.

• **RQ2: How robust are models trained with active learning?** We answer this question by measuring the theoretical robustness of the trained models (using the CLEVER score [102]) as well as their empirical robustness against multiple adversarial attacks. For the image classification task, our results reveal a gap between the fully trained models and those trained with active learning (up to 1.49 CLEVER score and 23.39% of success rate), whereas there are no such differences in the text classification task.

• **RQ3: Do models trained using active learning maintain accuracy after compression?** Model compression, a well-known technique to embed DL models in resource-constrained devices, has the downside effect of reducing the accuracy due to precision loss in the computed model weights. We investigate whether models trained with active learning are more sensitive to this phenomenon than fully trained models. Our results reveal that, for the image classification task, training on 50% of data entails a loss in test accuracy up to 7.47% higher compared to training with the full dataset.

• **RQ4: What is the relationship between the amount of training data, and model robustness and accuracy after compression?** We conduct additional experiments where we increase the data budget of data selection metrics. Our results indicate that with the growth of training data, the robustness of the model will also increase. The model can achieve similar robustness with the fully trained model only using 35% data, and even outperform it when more data are used. This indicates that the training process promoted by active learning can be used as an effective way to increase robustness. As for model compression, there appears to be no relationship between accuracy decay induced by compression and data budget.

With our extensive empirical study, we provide practical guidance to engineers in balancing the benefits of data selection metrics with their potential side effects. That is, we show and quantify the existence of a trade-off between the efficiency of model training (in particular, the data labeling effort) and model properties of interest (viz. robustness and accuracy after quantization). Doing so, we also open research directions to explore this trade-off and new data selection metrics aimed towards the different quality criteria.

In summary, the main contributions of this paper are:

- We conduct the largest empirical study that investigates the effectiveness of training data selection metrics on different classification tasks (image and text).
- Beyond (test) accuracy, we explore the effects of reducing the number of training data on the adversarial robustness of the models and their accuracy after compression. We, therefore, reveal the potential effects that active learning can have on these quality indicators.
- Thereby, we reveal a potential trade-off between labeling cost and the aforementioned quality indicators. This paves the way for future research in designing multi-objective data selection metrics aiming at optimizing this trade-off under a constrained labeling budget.

5.2 Overview

We first introduce the three-phase design of our empirical study, then present data selection metrics, datasets and models, and evaluation measures that are studied in our work.

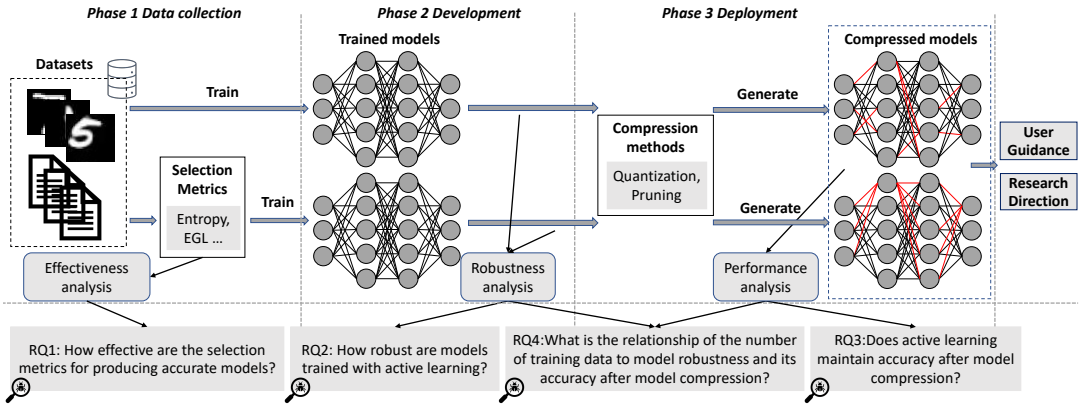


Figure 5.1: Overview of experiment design

The secure life cycle of deep learning is composed of some key stages [14] from the requirement analysis and data-label pair collection to the maintenance and evolution of the deep learning model. This paper studies the effect of the data collection (i.e., active learning) on the model development and deployment (i.e., the model quality). Figure 5.1 gives an overview of our study, which consists of three phases, effectiveness analysis of model training with different data selection metrics, adversarial robustness analysis of the trained model, and performance analysis after model deployment. Overall, we compare the models trained using the entire dataset and a subset of data selected by a specific data selection metric.

5.2.1 Study Design

More specifically, in the first phase (data collection), we compare the effectiveness of each data selection metric for training a model. Using different metrics (e.g., Entropy, Margin), we iteratively select a subset of training data and train the model, then observe the convergence trend of the test accuracy. For comparison, we also train two baseline models using entire training data and randomly selected data, respectively.

In the second phase (model development), we evaluate the robustness of the trained model. In our study, we apply multiple metrics (e.g., empirical robustness, CLEVER score) to compare the robustness of models trained with the selected and the model trained by the entire training data.

In the third phase, we focus on the model performance (test accuracy) in the model deployment phase. In general, a DNN model usually consists of a huge number of parameters, e.g., a VGG16 model requires about 258MB of hard disk memory. Before deploying such a large DNN model into the hardware (e.g., mobile devices), one has to consider the performance including the required memory and inference speed. We adopt two well-known techniques (model quantization and model pruning) to optimize a trained model. Then we evaluate the accuracy of the optimized models. Besides, we study the impact of training data size on the robustness of models and the accuracy of optimized models.

Finally, based on the results of our empirical study, we provide some practical guidelines for the usage of active learning on different tasks and summarize some potential research directions.

5.2.2 Datasets and DNN Models

We conduct experiments with two popular image datasets (MNIST [73] and CIFAR-10 [75]) and three widely used text datasets (IMDb [103], TagMyNews [104], and Yahoo! Answers [105]). MNIST includes 10-class grayscale images of hand-written digits. The dataset includes 60000 and 10000 training and test data, respectively. CIFAR-10 is a collection of 10-class color images (e.g., airplane, bird). The dataset consists of 50000 and 10000 training and test data, respectively. IMDb is a dataset including movie reviews widely used for text sentiment analysis (binary classification). Both the training and test sets include 25000 text reviews. TagMyNews provides news headlines (text) in 7 categories (e.g., Sport, Business). We randomly collect 20000 data for training and 2000 data for testing. Yahoo! Answers consists of text data of 10 topic categories (e.g., Society & Culture, and Science & Mathematics). We obtain this data from [106] directly with 3560 training data and 889 test data.

For each dataset, we employ two DNN architectures to reduce the model-dependent influence on the results. For MNIST, we use two well-known convolutional neural networks LeNet-1 and LeNet-5 [107], and for CIFAR-10, we select two models NiN [108] and VGG16 [109], both of which achieve high accuracy. For IMDb, TagMyNews, and Yahoo! Answers, we use two types of RNNs, LSTM, and GRU, derived from a base model [110]. Our companion website presents all details about the models and training parameters [72].

5.2.3 Data Selection Metrics

Various data selection metrics have been proposed and verified to reduce the labeling effort. Note that the data selection metric is also known as the acquisition function in the ML community. We include 14 data selection metrics from both the ML community (8 metrics) and the SE community (6 metrics). We first introduce the ones from the ML community.

• **Entropy** [89] considers the uncertainty of data using the prediction output. This metric is based on the Shannon entropy of the prediction probability:

$$\arg\max_{x \in X} \left(-\sum_{i=1}^{N} p_i(x) \log p_i(x) \right) \tag{5.1}$$

• **Margin** [89] computes a score for each data by the difference between its top-2 prediction probabilities:

$$Margin(x) = p_k(x) - p_j(x) \tag{5.2}$$

where $k = \arg\max_{i=1:N} (p_i(x))$ and $j = \arg\max_{i=\{1:N\}/k} (p_i(x))$. The training data with low scores will be selected for training.

• **K-center** [91] firstly divides data into K groups via some unsupervised machine learning methods (e.g., K-means clustering), then selects the center of each group (if not enough, consider the data that are close to the center). These selected data are regarded as the representative of the entire group.

• **Expected Gradient Length (EGL)** [31] assumes that the model has no knowledge of the true label of data in advance. For each data, it computes the expectation of the gradients by assigning all the labels to x. The data that have great expectations are selected. *EGL* can be presented as follows:

$$EGL\left(x\right)=\sum_{i=1}^{N}p_{i}\left(x\right)\left|\left|\nabla_{x}J\left(x,i\right)\right|\right| \tag{5.3}$$

where $||.||$ is the Euclidean norm. $\nabla_{x}J\left(x,i\right)$ is the gradient of the loss J given x and label i.

• **Bayesian Active Learning by Disagreement (BALD)** [111] applies dropout to select the most uncertain data:

$$\arg\max_{x\in X}\left(1-\frac{count\left(mode\left(y_{x}^{1},...,y_{x}^{T}\right)\right)}{T}\right) \tag{5.4}$$

where T is the number of applying dropout to the model.

• **Entropy-dropout** and **Margin-dropout** [90] apply dropout and calculate the average score of Entropy and Margin over all dropout models, e.g., Entropy-dropout is defined as

$$\arg\max_{x\in X}\frac{\sum_{i=1}^{T}Entropy\left(x^{i}\right)}{T} \tag{5.5}$$

where $Entropy\left(x^{i}\right)$ is the entropy of x at the i-th time.

• **Adversarial active learning (Adversarial_al)** [92] leverages an adversarial attack, DeepFool, to facilitate selecting data. Concretely, the model predicts labels for each data, then DeepFool introduces perturbations to each data until reaches an adversarial example. Finally, the data requiring smaller perturbations will be selected. Intuitively, such data are close to the decision boundary of the model.

Next, we introduce the metrics from the SE community.

• **Neuron Coverage (NC)** [16] selects data that have the largest neuron coverage:

$$\arg\max_{x\in X}\frac{\left|\left\{neu\right|\,neu\in Neurons\wedge activate\left(neu,x\right)\right\}\left|\right.}{\left|Neurons\right|} \tag{5.6}$$

where $activate\left(neu,x\right)$ indicates that the neuron neu is activated by x, namely, the output of neu is greater than a predefined threshold. We highlight that this is a variant of the original metric to fit active learning. The original NC aims at selecting data to cover all the neurons, however, in practice, just a few data are enough to reach 100% coverage [112]. Namely, after a few selection steps, all the neurons have been activated at least once and the marginal increase of coverage will be 0. Thus, we apply this variant to fit active learning.

• **K-Multisection Neuron Coverage (KMNC)** [17] improves NC by splits the lower and upper bounds of a neuron's output into k sections. Instead of using the coverage of neurons, it considers the coverage of sections. Similar to NC, the data with high coverage will be selected.

• **Multiple-Boundary Clustering and Prioritization (MCP)** [96] is an extension of Margin. First, it divides data into various "boundary areas" based on the top-2

predicted classes, e.g., for data with 10 classes, there are $P(10, 2) = 90$ permutations of pairwise classes. Next, MCP selects data from each area based on Margin.

• **DeepGini** [97] selects the most uncertainty data by:

$$\arg\max_{x \in X} \left(1 - \sum_{i=1}^{N} \left(p_i(x)\right)^2\right) \tag{5.7}$$

• **Likelihood-based Surprise Adequacy (LSA)** and **Distance-based Surprise Adequacy (DSA)** [95] measure the surprise adequacy by the dissimilarity between a test data and the training set. The difference between LSA and DSA is that LSA uses kernel density estimation to estimate the surprise adequacy, while DSA uses Euclidean distance. Both select data with the largest surprise adequacy.

We remark that although the initial purpose of these SE metrics is not for active learning, their underlying selection criteria share similarities with the criteria that active learning metrics rely on. For example, both Entropy (active learning metric) and DeepGini (SE metric) metrics select the uncertain data based on the output probabilities. Therefore, we believe that it is necessary to consider them in our study. Besides, no existing research has revealed that whether these SE metrics fit in active learning or how they perform.

5.2.4 Evaluation Metrics

Effectiveness. To evaluate the effectiveness of a selection metric, we use random selection as the baseline and calculate the difference of accuracy between the models trained using random selection and this metric. The difference is defined by:

$$diff = \sum_{i=1}^{steps} \left(Acc_{TS}^{i} - Acc_{TR}^{i}\right) \tag{5.8}$$

where *steps* is the total number of training steps. Acc_{TS}^{i} is the test accuracy of the model trained by a selection metric, and Acc_{TR}^{i} is by random selection. $diff$ shows the degree of a metric outperforming random selection.

Adversarial robustness. The robustness of DNNs refers to the ability to cope with adversarial examples. The robustness of DNNs can be evaluated in multiple ways. We introduce two popular methods of robustness estimation, Empirical Robustness [113] and CLEVER score [102]. **1) Empirical Robustness** This method quantifies the robustness of a DNN model through the success rate of crafting adversarial examples by an attack. In practice, given a DNN and a test set S, an attack attempts to craft an adversarial example based on each test data. The attack success rate is the ratio of adversarial examples successfully generated:

$$ASR = \frac{|\{x \mid x \in S \wedge y_{x'} \neq y\}|}{|S|} \tag{5.9}$$

Recall that $y_{x'}$ and y are the predicted label of x' and true label of x, respectively. A small ASR indicates a robust DNN. **2) CLEVER Score** The Cross-Lipschitz Extreme Value fornEtwork Robustness (CLEVER) score calculates the lower bound for crafting an adversarial example given an input, which is the least amount of perturbation required to fool a DNN model. A great CLEVER score indicates a robust DNN.

Table 5.1: Configurations of active learning

Dataset	Model	Step size	Stop point	Entire training size
MNIST	LeNet-1, LeNet-5	500	10000	60000
CIFAR-10	NiN, VGG16	2500	25000	50000
IMDb	LSTM, GRU	500	12500	25000
TagMyNews	LSTM, GRU	500	12500	25000
Yahoo!Answers	LSTM, GRU	500	12500	25000

Table 5.2: Configurations of adversarial attacks.

Dataset	FGSM	JSMA	C&W	CLEVER
	ϵ	γ	c	dis
MNIST	0.1, 0.2, 0.3	0.09, 0.1, 0.11	9, 10, 11	L-1, L-2, L-inf
CIFAR-10	0.01, 0.02, 0.03	0.01, 0.02, 0.03	0.1, 0.2, 0.3	

5.3 Implementation and Configuration

Experimental environment. This project is implemented based on Keras [114] and TensorFlow [59] frameworks. We run all experiments on a high-performance computer cluster except the model pruning and quantization. Each cluster node runs a 2.6 GHz Intel Xeon Gold 6132 CPU with an NVIDIA Tesla V100 16G SXM2 GPU. For the model pruning and quantization, we conduct the experiments on a MacBook Pro laptop with macOS Big Sur 11.0.1 with a 2GHz GHz Quad-Core Intel Core i5 CPU with 16GB RAM.

Active learning. We initialize an empty labeled pool and an unlabeled pool with all the unlabeled data. Besides, we initialize the with random weights. Next, in each training step, a fixed number (step size) of data is selected from the unlabeled pool by a specific metric. Then the selected data are merged into to labeled pool after annotation. The DNN is updated by retraining using all the data in the labeled pool. The procedure terminates when the size of the labeled pool reaches a threshold (stop point). Table 5.1 lists the detailed settings. Note that previous works have different parameter settings [115, 98, 89, 90], we balance these settings to set up our experiments. We implement data selection metrics based on [98].

Robustness. We use two public libraries, Foolbox [116] for empirical robustness evaluation and ART [117] for CLEVER score calculation. In empirical robustness, we apply three attack methods, i.e., FGSM, JSMA, and C&W, for the image classification task, and conduct three groups of experiments with different parameters as in [118]. For the text classification task, we use PWWS and DWB with the default setting in [106, 119] to attack the text-related models. By default, only the correctly classified data undertake the attacks. In the CLEVER score, the setting of c and dis follows the configuration in [118]. One difference is that we use 500 test data to calculate the CLEVER score, while [118] used 50. The detailed information is shown in Table 5.2. Note that the setting of CLEVER works both for the image and text classification tasks.

Model compression In model quantization, we apply two lightweight frameworks, CoreML [120] and TensorFlowLite [121], to transform DNNs into different bit-level versions. For CoreML, we use three levels, 2-bit, 4-bit, and 8-bit. For TensorflowLite, we apply 8-bit and 16-bit level quantizations in our models since it only supports these two levels. In model pruning, for both the weight- and neuron-level,

we prune a DNN into six compress versions with different degrees, from 10% to 60% at 10% intervals, using the implementations by [122, 123]. As a reminder, model pruning is conducted after the model is well-trained.

Last but not least, to reduce the influence of randomness, we repeat each experiment three times and compute the average results. In total, we trained and evaluated more than 2000 models in this study. The source code can be found on [2].

5.4 Experimental results

In this section, we present the results and answer each research question mentioned in Section 8.1. In the remaining parts, we remark that "TE", "TS", and "Random" represent training using the entire dataset, the subset selected by data selection metrics, and randomly selected data, respectively. Note that due to the page limitation, we only illustrate a part of the results in this paper, and the complete results are provided as supplementary material [72]. The conclusions we draw below generalize to all studied datasets/subjects and DNNs.

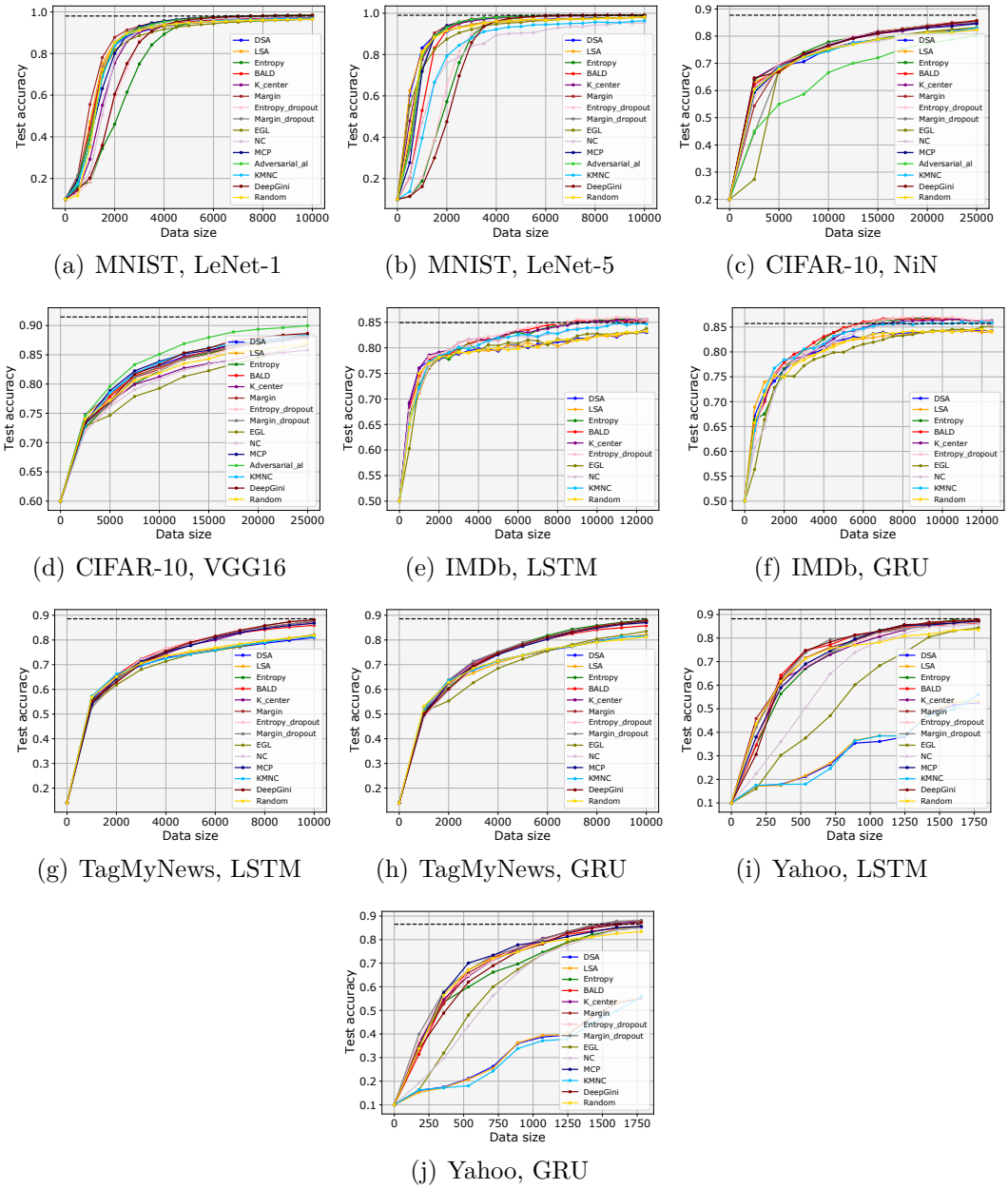
5.4.1 RQ1: Effectiveness of Data Selection Metrics

First, we show, in Figure 6.1, the performance of using different data selection metrics to train a DNN that achieves the same test accuracy as the fully trained model. For comparison, a horizontal dashed line in each sub-figure represents the accuracy of the fully trained model. Overall, most metrics manage to produce models with the same accuracy as the fully trained model by using only 7% to 50% of training data.

Next, Table 5.3 offers a more detailed comparison of these metrics, showing their effectiveness by taking random selection as a baseline (measured using Equation 5.8). Surprisingly, only three metrics (Margin, Margin-dropout, MCP) significantly outperform the baseline in all cases. On the contrary, in most cases (9 out of 10), EGL is worse than random selection. There are some metrics like LSA and DSA, which outperform the baseline on the image classification task but underperform the baseline on the text classification task, e.g., on Yahoo-LSTM and Yahoo-GRU, and the difference reaches up to 392.39. We conjecture that LSA and DSA tend to select data based on similarity or distance. However, different from image data, the two text data (sentence or document) might have a big difference in the hidden space even if they are in the same category. In this case, the inter-output of the model against these two data can be completely different which makes LSA and DSA select the wrong data.

Specifically, considering the image classification task, we observe that in addition to Margin, Margin-dropout, and MCP, two other metrics, LSA and DSA, can always outperform the baseline. What's more, on LeNet-1 and LeNet-5, half of the metrics are worse than the random baseline. Especially, Entropy, Entropy-dropout, NC, and DeepGini get high negative differences, e.g., -131.88 for Entropy on LeNet-1, which means these four metrics are much worse than random selection. Turn to sub-figures 5.2(a) and 5.2(b), in the first few training steps (where the big difference comes from), these four metrics achieve much less test accuracy than the others (also random selection). One explanation is that Entropy, Entropy-dropout, and DeepGini try to find the most uncertain data. However, such uncertain data are hard to be learned by models that are not well-trained.

[2] https://github.com/code4papers/ALempirical



(a) MNIST, LeNet-1 (b) MNIST, LeNet-5 (c) CIFAR-10, NiN

(d) CIFAR-10, VGG16 (e) IMDb, LSTM (f) IMDb, GRU

(g) TagMyNews, LSTM (h) TagMyNews, GRU (i) Yahoo, LSTM

(j) Yahoo, GRU

Figure 5.2: Evolution of the test accuracy (y-axis) achieved by different data selection metrics given the number (x-axis) of training data.

On the other hand, for the text classification task, the output probability-based metrics (e.g., Margin and BALD) perform better than the coverage and surprise adequacy-based metrics. Extremely, LSA performs worse than the baseline on 5 (out of 6) models. Besides, on model Yahoo_LSTM, the most efficient metric (Margin) is much better than the worst one (KMNC) with a 418.49% test accuracy gap. These results reflect that both coverage and surprise adequacy based-metrics are not suitable for the text classification task.

Answer to RQ1: The nature of the classification task significantly affects the effectiveness of multiple data selection metrics (e.g., LSA and DSA). Therefore, the limitation of active learning experiments to a single target task – even with multiple datasets – constitutes a critical threat to external validity. Some metrics, like Margin,

Table 5.3: Effectiveness with respect to random selection (baseline). The results above the baseline are highlighted in gray.

Metric	MNIST		CIFAR-10		IMDb		TagMyNews		Yahoo!Answers		Average
	LeNet-1	LeNet-5	NiN	VGG16	LSTM	GRU	LSTM	GRU	LSTM	GRU	
Entropy	-131.88	-183.84	26.14	3.91	49.41	44.13	29.63	37.72	3.19	-20.06	-14.16
Margin	69.82	28.78	14.44	6.29	-	-	29.68	36.23	26.10	17.44	28.59
K-center	-23.28	28.63	21.00	-7.67	62.94	39.15	25.60	36.63	9.52	15.97	20.84
EGL	-11.71	-14.13	-30.46	-17.65	3.62	-31.02	-14.55	-14.85	-155.46	-82.90	-36.91
BALD	27.84	-25.22	21.98	9.04	57.62	53.44	28.42	27.43	20.02	6.41	22.69
Entropy-dropout	-81.71	-165.96	22.88	3.18	64.81	48.91	30.23	34.43	2.29	17.51	-2.34
Margin-dropout	43.57	14.67	1.19	6.56	-	-	26.83	41.00	25.68	29.92	23.67
Adversarial _al	33.23	22.69	-77.73	29.15	-	-	-	-	-	-	1.835
NC	-26.95	-182.01	-2.76	-10.95	36.28	12.69	30.77	25.27	-72.29	-93.89	-28.38
KMNC	3.80	-153.02	0.40	12.69	34.02	39.39	-4.35	0.83	-392.01	-375.40	-83.36
MCP	25.66	8.39	14.28	13.17	-	-	24.53	30.70	11.59	17.02	18.16
DeepGini	-74.07	-213.23	21.33	12.98	-	-	30.62	31.98	20.47	-2.59	-21.56
LSA	19.83	22.37	3.41	8.97	-4.53	5.10	-1.22	-1.32	-385.71	-359.77	-69.28
DSA	9.26	23.62	1.37	10.49	5.54	5.34	-7.87	-0.78	-392.39	-359.47	-70.48

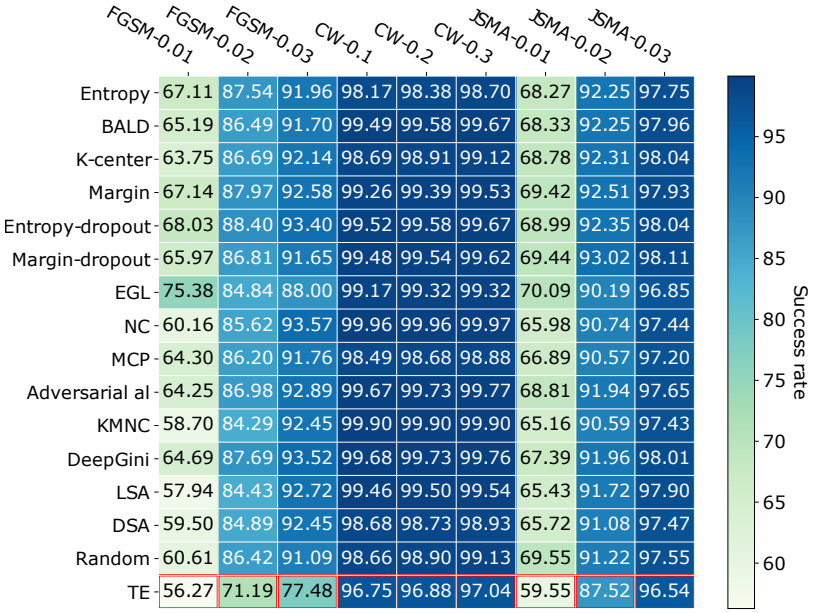
Margin-dropout, and MCP, consistently perform well across all labeling budgets, models, and tasks.

5.4.2 RQ2: Adversarial Robustness

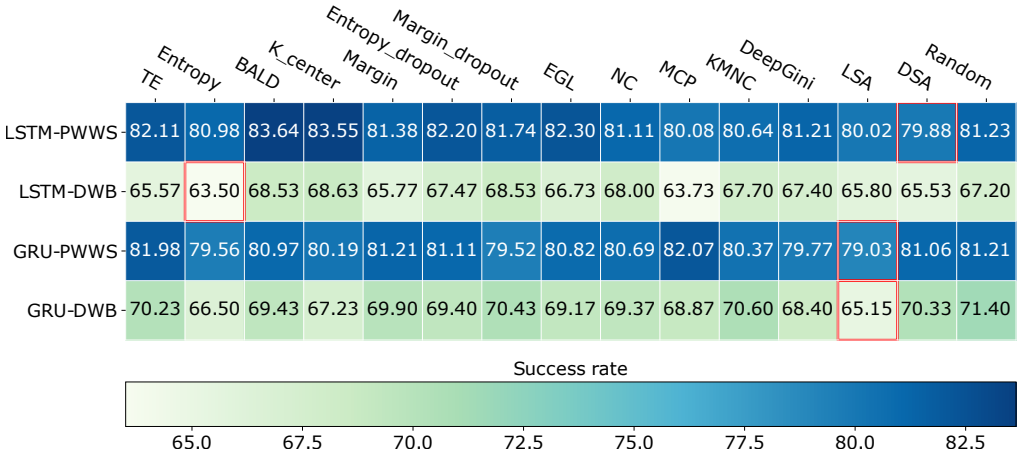
We then study the robustness of models trained by different data selection metrics against adversarial attacks. Figure 5.3 illustrates the empirical robustness of the models against various attacks. Once again, we have to distinguish the two types of tasks since the results of different metrics are greatly biased on the tasks. For the image classification task, the TE models are usually more robust than the TS models. However, for the text classification task, the TS models are in some cases more robust than the TE models (ad vice-versa). We conjecture that two factors may affect the robustness of the models: (i) the number of data used for training and (ii) the training process. In active learning, the early selected data are trained more times during incremental learning. Thus, the most informative data (according to the data selection metrics) have a larger influence on the model weights and, in turn, impact its robustness. On the other hand, since the selected data can be representative of the entire dataset to some extent, the difference between the TS and TE models is small. Taking VGG16 as an example, the difference varies from 1.67% to 17.21% in FGSM, from 1.42% to 3.21% in C&W, and from 0.32% to 10.54% in JSMA. Another observation that reinforces our hypothesis regarding the importance of the training process is that none of the metrics performs consistently better than random selection. We investigate this hypothesis in the RQ4 experiments, where we consider different labeling budgets.

Table 5.4 lists the CLEVER score of different models. As for the text classification task, models trained with active learning either yield a small improvement (less than 0.58 CLEVER score) or offer inconsistent benefit (either increasing or decreasing the CLEVER score, depending on the considered metric and norm distance). Besides, none of the data selection metrics improves over the random selection, even the three metrics which performed better in terms of effectiveness (viz. Margin, Margin-dropout, and MCP). Overall, these results corroborate our previous findings. That is, for the image classification task, active learning yields less robust models.

Answer to RQ2: For the image classification task, models trained with active learning can have lower robustness than models trained with the entire dataset, and the difference can be up to 1.49 CLEVER score and 23.39% attack success



(a) CIFAR-10, VGG16



(b) TagMyNews

Figure 5.3: Adversarial attack success rate (%) of VGG16 and TagMyNews. The number in each cell represents the success rate and the color gives a straightforward visual comparison of different values. The most robust model is framed by a red rectangle. The lower the success rate, the better robustness.

rate. Therefore, experimental studies should involve evaluation metrics beyond clean accuracy. For the text classification task, the results are inconsistent across attacks (empirical robustness) and distance norm (CLEVER score), indicating that other factors are at play when it comes to robustness, e.g., the training process.

5.4.3 RQ3: Test Accuracy After Model Compression

We compare the test accuracy of a model produced by different data selection metrics before and after compression. Table 5.5 shows the results of VGG16 (image classification) and TagMyNews-LSTM (text classification) by quantization.

On VGG16, the accuracy of the 2-bit compressed models drops significantly by at least 76.32%, no matter it was fully trained or with active learning. By contrast, on TagMyNews-LSTM, the accuracy decay using 2-bit is relatively small (less than

Table 5.4: The CLEVER score of CIFAR-10 (VGG16) and TagMyNews (LSTM). The results that are better than the TE model are highlighted in gray. The higher score, the better robustness.

Meric	CIFAR-10 (VGG16)			TagMyNews (LSTM)		
	L-1	L-2	L-inf	L-1	L-2	L-inf
Etropy	3.7143	0.3289	0.0055	3.6466	1.1559	0.0531
Margin	3.6607	0.2136	0.0051	3.5496	2.0390	1.6712
K-center	3.6475	0.3132	0.0057	3.7970	0.9840	0.0132
EGL	4.2577	0.6996	0.0037	3.6881	0.7235	0.0276
BALD	3.8096	0.3176	0.0048	3.7424	1.3236	2.9914
Etropy-dropout	3.6881	0.1495	0.0075	3.4033	0.7465	0.4027
Margin-dropout	3.8213	0.3103	0.0047	3.5453	0.8562	0.0059
Adversarial_al	4.2606	0.5242	0.0032	-	-	-
NC	4.2036	0.7017	0.0031	3.9315	0.9729	0.1879
KMNC	4.2628	0.6442	0.0340	3.1824	0.4198	0.0041
MCP	4.2955	0.4984	0.0041	3.7244	1.1289	0.8011
DeepGini	4.2616	0.5116	0.0031	3.4139	2.6286	1.6805
LSA	4.2449	0.7477	0.0041	3.8102	0.7150	0.0077
DSA	4.2567	0.6576	0.0035	3.7198	1.0144	0.0102
Random	3.9665	0.6499	0.0057	3.6295	1.6243	0.0065
TE	4.3636	0.7399	0.8053	3.3600	0.9910	1.6684

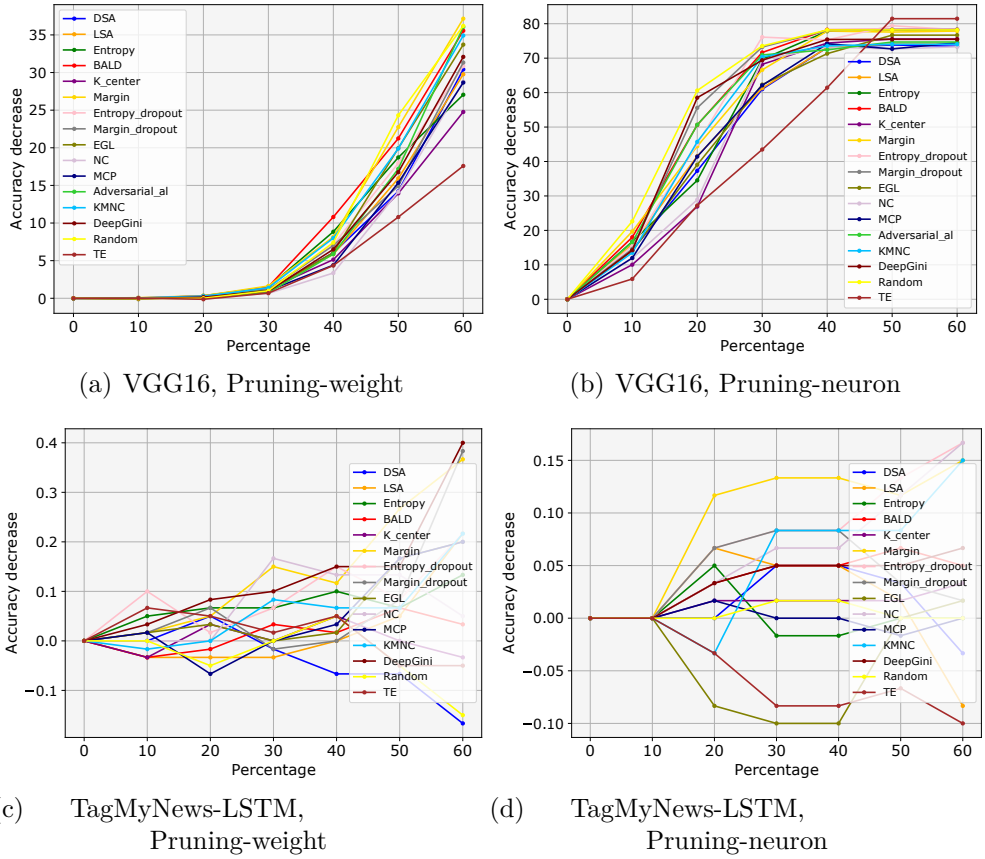
9.42%). The reason could be that, *1)* the quantization process has less impact on the LSTM layer, or *2)* the text data is less sensitive to the precision of weights. In both cases, the compressed models achieve the same accuracy as the original models with 16-bit (expect Margin and Entropy-dropout On TagMyNews-LSTM). Specifically, compared with random selection, on both VGG16 and TagMyNews-LSTM, no metric always outperforms the baseline. Comparing with the TE model, we found that for the image classification task, the compressed TE model always maintains higher (with a gap up to 7.47%) test accuracy. However, for the text classification task, the TS model sometimes loses more test accuracy than the TS models after quantization. Surprisingly, with 2-bit quantization, the TE model gets the largest accuracy decay (-9.42%).

Table 5.5: The change of test accuracy of CIFAR-10 (VGG16) and TagMyNews (LSTM) before and after model quantization. The best and worst results are highlighted in gray and orange, respectively.

	CIFAR-10 (VGG16)					TagMyNews (LSTM)			
	CoreML			TFLite		CoreML			TFLite
	2-bit	4-bit	8-bit	8-bit	16-bit	2-bit	4-bit	8-bit	16-bit
Entropy	-78.13	-4.77	-0.09	-0.53	0	-2.13	-0.2	0	0
Margin	-78.59	-3.06	-0.14	-0.76	0	-2.5	-0.28	-0.03	-0.02
K-center	-76.32	-2.28	-0.81	-5.18	0	-1.72	-0.1	+0.02	0
EGL	-76.72	-2.35	-0.02	-0.63	0	-0.22	-0.18	+0.12	0
BALD	-77.99	-2.34	-0.01	-0.2	0	-3.9	-0.03	+0.02	0
Etropy-dropout	-77.92	-4.53	-0.76	-0.48	0	-2.82	-0.1	-0.03	-0.02
Margin-dropout	-78.02	-1.56	-0.01	-0.39	0	-2.57	-0.22	-0.02	0
Adversarial_al	-79.77	-2.05	-0.05	-3.2	0	-	-	-	-
NC	-77.17	-8.85	-0.22	-2.39	0	-2.57	-0.12	+0.02	0
KMNC	-78.92	-1.65	-0.06	-2.67	0	-2.88	-0.22	+0.03	0
MCP	-79.72	-1.56	-0.02	-3.64	0	-4.17	-0.07	0	0
DeepGini	-80.2	-3.47	0	-3.02	0	-8.73	-0.13	-0.03	0
LSA	-78.16	-1.74	-0.01	-2.53	0	-4.1	-0.12	-0.02	0
DSA	-78.28	-1.99	-0.04	-2.88	0	-2.88	0	+0.02	0
Random	-77.82	-1.74	-0.24	-1.77	0	-2.32	-0.42	-0.13	0
TE	-81.36	-1.38	0	-0.03	0	-9.42	-0.07	0	0

Figure 6.2 depicts the result by weight- and neuron-level pruning. In general, with pruning more weights and neurons, the test accuracy decreases gradually up to 36.20% and 81.46% on VGG16, respectively. However, on TagMyNews-LSTM, the accuracy changes negligibly by increasing or decreasing up to 0.4%. The reason might be that these two pruning methods can only affect the Convolutional layer and the

Dense layer, while our LSTM models only contain one Dense layer to output the final prediction probability. Looking into VGG16, the neuron-level pruning affects the accuracy more than the weight-level for all the data selection metrics, which suggests that in practical applications, the engineers should consider more the weight-level pruning than the neuron-level to minimize the accuracy loss. Among these data selection metrics, DeepGini and NC are always better than random selection. Besides, for the image classification task, the TE model outperforms all TS models with a gap of up to 33.4%. However, for the text classification task, the TE model has no advantage of maintaining test accuracy over TS models after pruning.



(a) VGG16, Pruning-weight

(b) VGG16, Pruning-neuron

(c) TagMyNews-LSTM, Pruning-weight

(d) TagMyNews-LSTM, Pruning-neuron

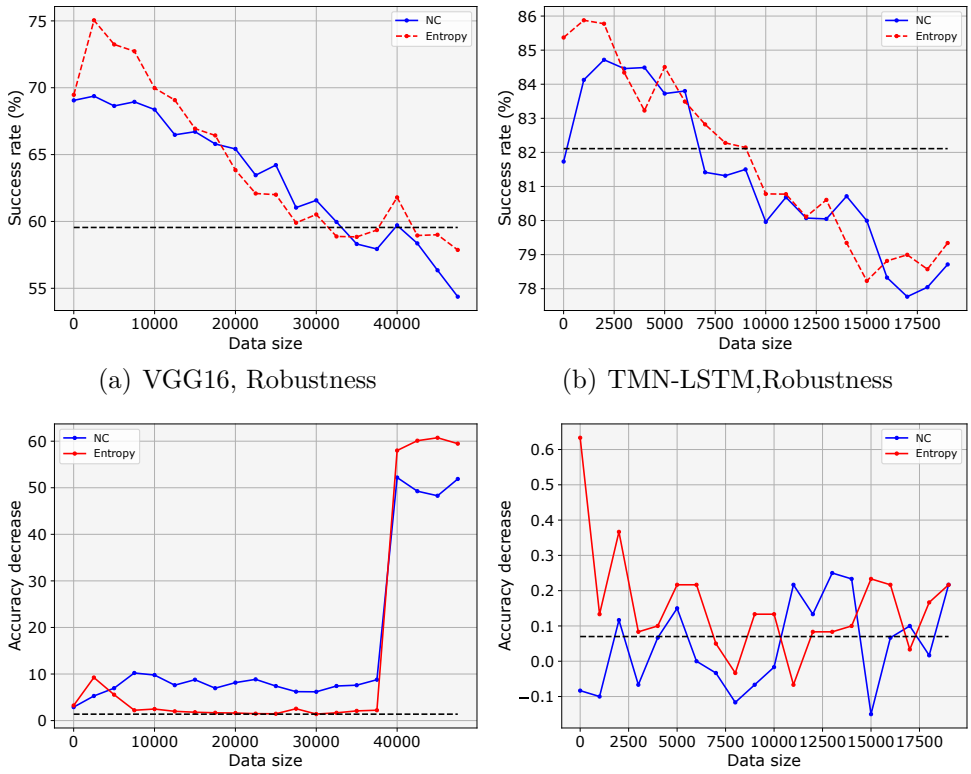
Figure 5.4: The change of test accuracy (y-axis) after model pruning with different degrees (x-axis), i.e. the proportion of weights and neurons being pruned.

Answer to RQ3: Model compression inconsistently affects the performance of the models trained with active learning and no data selection metric provides satisfactory results across all tasks. For the image classification task, after compression, the fully trained models hold higher test accuracy than the models trained with active learning, and the gap can be up to 7.47% (quantization) and 33.4% (pruning). On the contrary, for the text classification task, the fully trained models have no advantage of test accuracy over the models trained with active learning after model compression.

5.4.4 RQ4: Impact of Training Data Size

From Sections 5.4.2 and 5.4.3, we found that the data selection metrics tend to produce less robust models and introduce greater changes of accuracy after model compression. We further extend our experiments to investigate the impact of the data

size on the quality (e.g., adversarial robustness and test accuracy after compression) of models. Figure 5.5 shows the results on VGG16 and TagMyNews-LSTM. For adversarial attacks, we use JSMA-0.01 for VGG16 and PWWS for TagMyNews-LSTM, respectively. For model compression, we employ the 4-bit quantization by CoreML. On both models, we show the results by two data selection metrics: the prediction probability-based Entropy and neuron coverage (NC).



(a) VGG16, Robustness

(b) TMN-LSTM,Robustness

(c) VGG16, Quantization

(d) TMN-LSTM,Quantization

Figure 5.5: Impact of training data size (x-axis) on the performance (y-axis) (success rate or accuracy decrease) of model. The horizontal dashed line shows the attack success rate or change of test accuracy of a fully trained model. TMG–TagMyNews.

Figure 5.5(a) and Figure 5.5(b) show that TS models become more robust with more training data added, especially, the models could be more robust than the fully trained models in the end. According to the results, we can see that to achieve similar robustness with fully trained models, more than 60% of training data are required for the image classification task, while only 35% of training data is enough for the text classification task. This corroborates our hypothesis (see RQ2) that the training process used by active learning (which makes data selected earlier go through more training iterations than data selected later) can yield more robust models compared to a traditional training process involving all data from the beginning. This finding also opens the perspective of using such an active learning process in conjunction with common methods to improve robustness, such as adversarial training.

Figure 5.5(c) and Figure 5.5(d) show the test accuracy decay after model compression. For the text classification task, the difference is negligible throughout the increase in data size. For the image classification task, the decay of accuracy keeps stably lower than 10% until the data size reaches 37500, where the accuracy decreases significantly by up to 60.73%. These results reveal that the data size is not a key

factor in the test accuracy decay on model compression.

Answer to RQ4: Increasing the labeling budget of active learning mitigates the loss in robustness compared to a fully trained model. For example, using at least 60% training data for the image classification task (respectively, 35% for the text classification task) yields models with the same robustness as the fully trained model. However, training with more data does not change our previous conclusions regarding the inconsistent effect of model compression.

5.4.5 Discussion

We highlight our novel findings first, then discuss some practical guidance and research directions accordingly.

Novel findings and user guidance. 1) Finding: previous studies were limited to test accuracy, the image classification task, and did not compare metrics from both the ML and SE communities. We reveal that the benefits of active learning are highly task-dependent. Some data selection metrics (LSA and DSA) are highly affected by the nature of classification tasks, while some (Margin, Margin-dropout, and MCP) can achieve consistently high performance. Guidance: engineers need to choose data selection metrics according to specific tasks. **2)** Finding: the limitation of active learning also exists in the adversarial robustness and model compression, however, the evaluation was missing in the literature. We found that the active learning process has some potential but notable impact on these indicators, e.g., for the image classification task, the fully trained model is more robust than the model trained by active learning. Guidance: engineers are recommended to consider all these indicators when using active learning.

Research directions. 1) Since no existing data selection metric can perform well on all the considered objectives, an interesting research direction is to design data selection metrics that can optimize multiple qualities such as accuracy, robustness, and accuracy after compression. Regarding robustness, an adequate metric should also effectively integrate with an adversarial training process, minimizing the amount of data to form which adversarial examples are generated to increase the model robustness. **2)** Our study focuses on two types of classification tasks (image and text). Recently, DL systems have been applied in some SE-related tasks, such as source code function prediction and automatic program repair. Exploring how existing data selection metrics perform on these tasks is a potential research direction. Besides, proposing a source code-oriented data selection metric for this kind of system could be another contribution.

5.4.6 Threats to Validity

First, threats to validity may lie in the selected datasets and DNN models. Regarding the datasets, we employ five popular datasets across both image and text classification tasks in our study. As for the DNNs, we consider, in total, ten architectures (two for each dataset) to alleviate the model-dependent issue. Though the models we use perform well on the chosen tasks, an interesting direction of future work is to repeat the study on more complex model architectures, such as transformer-based models for text classification [124], and observe whether the trends remain.

Second, the parameter configuration may induce a threat to validity. Regarding the parameters in active learning, there is no universal rule for choosing the best

settings. For instance, previous studies [115, 98, 89, 90] utilize different settings of labeling budget and stop strategy. We mitigate this threat to validity in two ways: (1) we took the best and most common practices from the literature to design our active learning process and settings, and (2) our research questions concern the specific impact of some parameters (e.g. RQ4 studies the impact of the stop criteria). For robustness evaluation, we follow a recent study [118] to set a range of perturbation sizes for different adversarial attacks. As for model compression, our study involves multiple settings to reduce the potential bias of the results.

Third, due to the space limitation, we only report the results of two datasets covering the image and text classification tasks. Nonetheless, we remark that the reported conclusion is generalized to all the datasets and DNNs. For example, for RQ2, in total, 31 out of 36 settings, the fully trained image-related models are more robust than the actively trained models, which is consistent with our findings. Besides, we report all our complete results on our project site [72].

5.5 Conclusion

We conducted a comprehensive empirical study to explore the limitations of active learning. In total, more than 2000 models for image classification and text classification tasks have been trained and systematically evaluated. The results reveal that, when using active learning to train a model, different data selection metrics yield models of significantly different quality (in accuracy and robustness). For the image classification task, a model trained with active learning can achieve competitive test accuracy but suffers from robustness loss and has lower accuracy after compression. However, these downsides rarely occur in text classification models. Also, we further studied the relationship between the data budget and the quality of a trained model. We found that the robustness of the model increases with the amount of training data, and ultimately reaches the robustness of the fully trained model. Based on these findings, we provided some practical guidance as well as research directions. We believe that our work could give engineers and researchers some valuable insights into the whole secure life cycle of deep learning, especially in the data collection and model evolution steps.

6 Active Code Learning: Benchmarking Sample-Efficient Training of Code Models

In this chapter, we build the first benchmark to study sample-efficient training of code models – active code learning. Specifically, we collect 11 acquisition functions (which are used for data selection in active learning) from existing works and adapt them for code-related tasks. Furthermore, we explore future directions of active code learning with an exploratory study. We propose to replace distance calculation methods with evaluation metrics and find a correlation between these evaluation-based distance methods and the performance of code models.

Contents

6.1 Introduction

Using machine learning (ML) to help developers solve software problems (ML4Code) [125] has been a hot direction in both software engineering (SE) and ML communities in recent years. Deep learning (DL), one of the advanced ML techniques, has achieved great success in multiple software tasks, such as code summarization [126], code clone detection [127], and vulnerability detection [70]. Typically, preparing a code model involves two main steps: first, building a pre-trained model that learns general code information; second, fine-tuning this model using datasets that target a specific downstream task. Both components contribute to the success of ML4Code and are still under exploration for further improving the performance of code models.

Generally, pre-trained code models can be easily accessed from open resources, e.g., Hugging Face [60], or built by using self-supervised learning without data labeling effort. This means that for developers planning to use code models, the first step of pre-trained model preparation is not challenging and can be fully automated. However, collecting datasets to fine-tune pre-trained models is not easy. The main reason is that the general fine-tuning process follows the procedure of fully-supervised learning, which requires carefully labeled training data. Unfortunately, the data labeling process is time-consuming and labor-intensive [70].

To alleviate the aforementioned heavy effort of training data labeling, active learning [128] is used to enable sample-efficient model training in other famous fields, e.g., computer vision [129] and natural language processing [31]. The key idea of active learning is to iteratively select a subset of training data to label and use them to train the model. The existing studies [8] have shown that labeling only a few (less than 10%) training data can train a model with similar performance as the model trained by using the entire training data. In this way, the labeling effort can be significantly reduced and made flexible to a fixed budget. However, despite being well-studied in many application domains, the usefulness of active learning in ML4Code is still unknown. Researchers mainly focus on designing new model architectures, proposing novel code representation methods, and studying more software tasks. The study of how to lighten the model training cost is missed. There is a need to provide a benchmark to support the exploration of this important problem.

In this paper, we aim to bridge this gap and build a benchmark to study how active learning can help us efficiently build code models – active code learning. We collect acquisition functions (i.e., active learning methods) that can be used for code data from existing studies [8, 130]. In total, we implement 11 acquisition functions (including random selection) that can be divided into two groups, output-uncertainty-based functions (5 functions) and clustering-based functions (5 functions) in our benchmark. Then, based on these collected acquisition functions, we conduct experiments on four datasets (including classification tasks and non-classification tasks) and two famous pre-trained code models to answer the following research questions:

RQ1: What features should be used for clustering-based acquisition functions? Feature selection [131] is an important problem for clustering methods. However, it is unclear which features should be used for clustering-based methods in active code learning. Our first study is to explore how different features affect the effectiveness of active code learning. Here, we consider three types of features, code

tokens, code embedding vectors, and model output vectors. *Findings:* The results show that clustering-based methods are sensitive to the used features. Interestingly, in more than half of cases (62.5%), output vectors based methods achieve significantly better results than code token and code embedding-based methods.

RQ2: How do acquisition functions perform on code models? After determining features to use, we compare all the acquisition functions across different tasks. *Findings:* Firstly, contrary to the previous study [130], which suggests that simple techniques perform better for active learning, we found that clustering-based methods consistently outperform simple uncertainty methods in our considered binary classification code task (Clone detection). Secondly, unlike prior research [8, 130], which shows that only a small set of training data (less than 10%) is sufficient to produce good models, our results on code summarization tasks indicate that existing methods are not yet capable of achieving this goal. There is at least a 30% performance gap between the models trained by active learning (with 10% data) and the models trained using the entire training data. In total, in the first two studies, we trained 5200 models considering each labeling budget. The training process takes more than 3200 GPU hours.

Exploratory study. Additionally, using our benchmark, we conduct a study to further explore potential directions for proposing new acquisition functions. We focus on clustering-based methods since existing methods can not perform well on non-classification tasks. Concretely, due to clustering-based methods tending to select diverse data (data have a bigger distance to each other), we first check if there is a correlation between the distance of selected data and the accuracy of the trained model. Then, we propose a novel view to consider the distance of code for active learning – *using evaluation metrics (e.g., CodeBERTScore) as distance calculation methods*. Based on the evaluation, we found that, for non-classification code tasks, 1) there is no correlation between the distance (calculated by Cosine similarity and Euclidean distance) of selected data to each other and the accuracy of models. 2) There is a weak correlation between the evaluation metrics-based distance of selected data to each other and the accuracy of models, indicating that future proposed methods can be based on evaluation metrics-based distance.

To summarize, the main contributions of this paper are:

- To the best of our knowledge, this is the first work that builds a benchmark for sample-efficient training for code models. The source code as well as the datasets can be found on our project site [72].
- Based on our benchmark and empirical study, we found that multiple findings from previous works [8, 130] on image and text data are inapplicable to code data.
- We design a new strategy that uses evaluation metrics to compute the distance between code pairs to support future research when proposing new acquisition functions.

6.2 Benchmark

After collecting massive acquisition functions whose effectiveness on image data and text data has already been demonstrated in previous studies, we plan to study their unknown use case – if they are still useful for efficient training models of code. As mentioned in Section 1, we have two main questions want to answer, 1) since features used for clustering-based acquisition functions are important and highly related to the final performance of active learning, we explore four types of features for

each acquisition function, and check which kind of feature is suitable for conducting code selection. 2) As the previous study [130] claimed that – simple methods perform well on active learning, we want to check if this conclusion still holds on active code learning. Concretely, we compare output-uncertainty-based functions (i.e., simple methods mentioned in [130]) with clustering-based functions with carefully chosen features.

6.2.1 Study Design

To address the question of suitable feature selection (**RQ1**), we first prepare different versions of acquisition functions based on the features they use. Then, we conduct active learning on different code tasks using these functions and compare the performance of trained models. we can draw conclusions about which features are most suitable for each acquisition function. Here, we focus on three types of features from different perspectives, 1) code embeddings, 2) sequence of code tokens, and 3)model output vectors.

- **Code Embeddings** It is common to consider code embeddings as the input of clustering methods since code embeddings produced by pre-trained code models can present the general information of code. Code embedding is similar to the image data after image pre-processing. In our study, code embeddings extracted from the fine-tuned code models which can better represent the downstream task are used as the features.
- **Sequence of code tokens** Regardless of the code representation techniques, the raw program will be converted into a sequence of integer values. Thus, it is also possible to use these sequences as the inputs of the clustering methods. It is similar to using the pixel numbers of images as inputs. The only difference is that code tokens are often with bigger data space, i.e., the image pixels are from 0 to 255, while the range of code tokens depends on the vocabulary size, which is generally much bigger than 255, e.g., the vocabulary size of our used CodeBERT model is 50265.
- **Model outputs** Different from the above two features that have been widely explored in other fields and can be easily considered for active code learning, in our study, we propose to use the model outputs as the input features of clustering methods. The intuition behind this idea is that the output can be seen as the *understanding* of the model on this input data which should be useful for data selection. Specifically, for classification tasks, we use the one-hot output probabilities as the features, and for the non-classification tasks, the output vectors produced by decoders are used as the features.

For the comparison of each acquisition function (**RQ2**), the main goal is to find the recommended function that has consistently better performance than others in active code learning. At the same time, we also compare the output-uncertainty-based functions and clustering-based functions to determine if the previous findings [130] hold true in active code learning. Here, based on the first study, we use suitable features as the input for clustering-based acquisition functions and perform code selection.

6.2.2 Dataset and Model

Table 6.1 presents the datasets and models we used in the study. We consider three code tasks including a multi-class classification task (problem classification),

Table 6.1: Details of datasets and models. Accuracy (%) for problem classification, F1-score (%) for clone detection, and PPL/BLEU for code summarization.

Task	Dataset	Language	Train/Dev/Test	CodeBERT	GraphCodeBERT
Problem classification	Java250	Java	62500/-/12500	98.10	98.49
Clone detection	BigCloneBench	Java	90102/4000/4000	97.15	97.05
Code Summarization	CodeSearchNet	JavaScript	58025/3885/3291	3.85/14.34	3.79/14.89
	CodeSearchNet	Ruby	24927/1400/1261	4.04/12.80	3.99/13.54

a binary classification task (clone detection), and a non-classification task (code summarization).

- **Problem classification** is a multi-class classification task. Given a program, the model predicts the target problem that the program solves. We use the dataset JAVA250 [132] provided by IBM for this task. JAVA250 contains 250 code problems, e.g., problem: *write a program that prints the heights of the top three mountains in descending order.*
- **Clone detection** is a well-studied task in the software engineering field to lighten the effort of software maintenance. The main purpose of this task is to check if two programs are semantically equivalent. Thus, code clone detection is often seen as a binary classification problem. In our study, we use the dataset provided by BigCloneBench [133] which contains a large number of Java code clone pairs. Besides, for the computation friendly, we follow the previous work [134] and only use a subset of data from BigCloneBench.
- **Code summarization** is a common code task for helping developers understand code snippets. Given a program, the model generates natural language comments to describe the functionality of this program. We use two datasets provided by Microsoft [135] in our study.

For the code models, we focus on pre-trained code models since they achieved much better results than models trained from scratch [136, 137]. We follow the previous work [134] and use two well-known pre-trained code models in our study, CodeBERT [136] and GraphCodeBERT [137]. Other models can be easily added and evaluated to our provided project. Considering the downstream tasks, the pre-trained model is used as an encoder to generate code embeddings, and a decoder is followed to produce the final results for a specific code problem. For example, a dense layer is used as the decoder to produce the output probabilities of classification tasks. The same as the original works or CodeBERT and GraphCodeBERT, during the fine-tuning, we also fine-tune the parameters of pre-trained encoders to ensure a better performance on downstream tasks. Then, we briefly introduce the two pre-trained models here, and the detailed model information can be found on our project site [72].

- **CodeBERT** is a bimodal model that shares a similar architecture with the well-known BERT model in the field of natural language processing. The CodeBERT model is initialized through pre-training with six code datasets featuring various programming languages such as Java and Python. During the training process, programs are transformed into sequences of tokens, which is similar to text data. As a result, CodeBERT is able to learn the semantic information of code data.
- **GraphCodeBERT** is a newer pre-trained code model that incorporates both sequences of tokens and data-flow information to learn code knowledge. This allows pre-trained code models to understand the structural information of programs

and generate more precise code representations. In our study, the base model is learned by this combination of code information and fine-tuned for our considered downstream tasks only using the code token information. The reason is that we found in some downstream tasks, adding the data-flow information to fine-tune the model can have a negative influence on the performance of models, e.g., for the problem classification task, adding data-flow information and without data-flow information, the accuracy of fine-tuned models are 82.30% and 98.39%, respectively.

6.2.3 Evaluation Metrics

Our study contains three types of code tasks. For each task, we use the most practical metrics to evaluate the trained models.

Accuracy on the test data is the basic way to evaluate the performance of multi-class classification models. It calculates the percentage (%) of correctly classified data over the entire input data.

F1-score is a commonly used metric for binary classification problems. It calculates the harmonic mean of the precision and recall scores. Given that, true positive (TP) represents the number of samples correctly predicted as positive, false positive (FP) represents the number of samples wrongly predicted as positive, and false negative (FN) represents the number of samples wrongly predicted as negative, F1-score is calculated as:

$$F1 - score = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Perplexity (PPL) is a widely used metric to evaluate language models. PPL can be seen as the loss of language models that can record the logs of the training process. A lower PPL score means a better performance of the model. Recently, researchers applied PPL to record to evaluate the code summarization models [138]. Specifically, PPL is calculated by:

$$PPL\,(X) = \exp\{-\frac{1}{t}\sum_{i=0}^{t}\log p_\theta(x_i|x_{<i})\}$$

where $X = (x_0,\ x_1,...,x_t)$ is the set of code tokens, and $logp_\theta(x_i|x_{<i})$ is the log-likelihood of the ith token conditioned on the preceding tokens $x_{<i}$ by the model.

BLEU (Bilingual Evaluation Understudy) is a metric to evaluate the quality of the generated text to another (the reference). Simply, the BLEU score is calculated by:

$$BLEU = BP \times \exp\sum_{n=1}^{N} w_n \log p_n$$

The value of N depends on the used N-gram precision, and the weights w_n = N / 4. In our study, 4 is used. p_n is the ratio of length n sub-sequences in both the candidate sequence and the reference. BP is the brevity penalty calculated by:

$$BP = \begin{cases} 1, & if c > r \\ e^{1-r/c}, & if c \leq r \end{cases}$$

where c is the length of the generated sequence and r is the length of the reference sequence.

Besides, to perform a significance test for the comparison between different acquisition functions, we conduct statistical analysis by using Student's t-test [139] in our study.

6.2.4 Configurations

Training configuration. Considering the hyperparameters used for model training, we follow the previous work [134] and use the same batch size and learning rate for each model. All the detailed settings can be found on our project site. For model fine-tuning, we set the training epoch as 10 which is enough to ensure the convergence of models.

Active learning configuration. We initialize the models (which is a common setting of active learning that we start from a model with a little knowledge of the datasets) by training them on randomly selected 500 samples. We set the labeling budget as 1% of the entire training set and do 10 times iterations of active learning.

Acquisition function configuration. For all clustering-based methods, we follow the previous work [8] and set the number of centers as the labeling budget. For BALD, we set the times of dropout prediction as 20. For all clustering-based methods, to get more precise code information, we obtain the embedding features from the fine-tuned encoders.

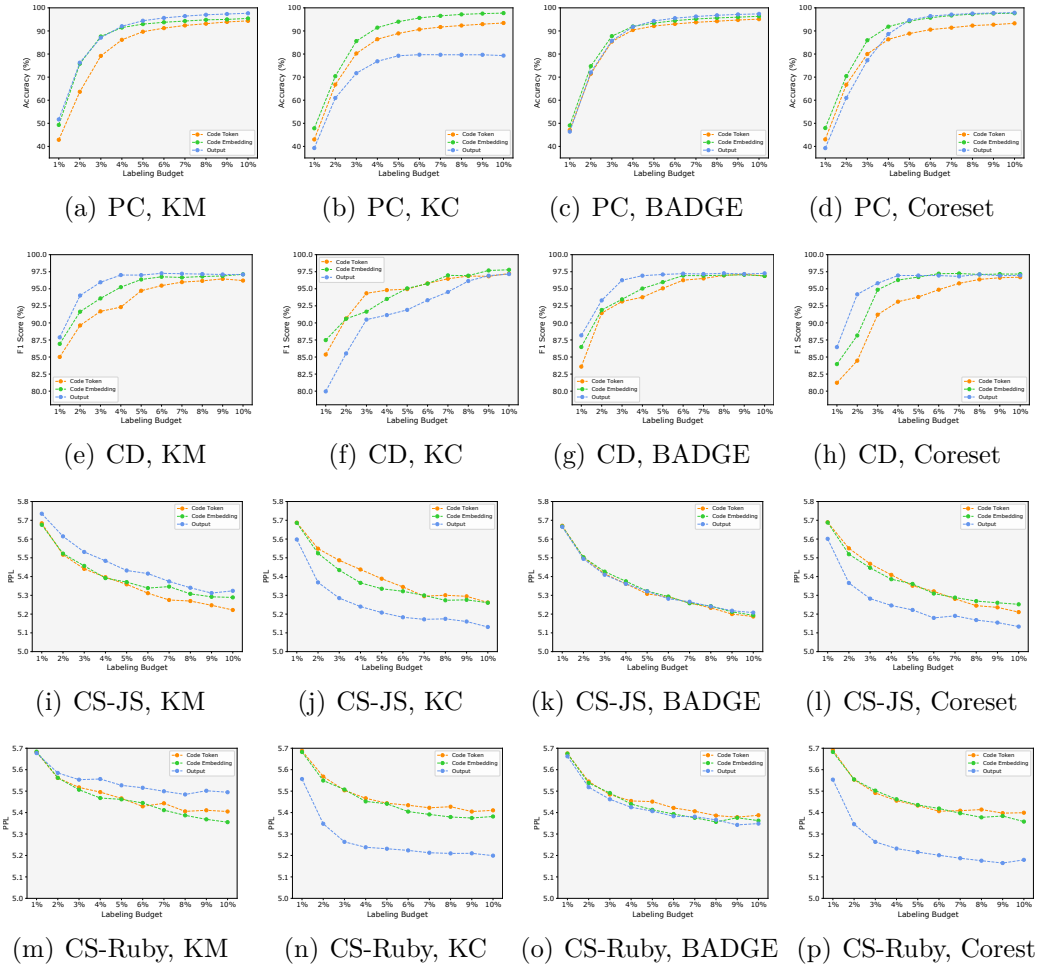
6.2.5 Implementation and Environment

The project is based on Python-3.6 and PyTorch-1.10.2 framework. The key implementation of code models is modified from the open source project [135]. We adopt acquisition functions implemented by [8, 130] that are used for image data and text data to code data. All the source code can be found on our project site. We conduct all the experiments on a 2.6 GHz Intel Xeon Gold 6132 CPU with an NVIDIA Tesla V100 16G SXM2 GPU. We repeat all the experiments five times to reduce the influence of randomness and report the average results in the following sections.

6.2.6 RQ1: Feature Selection

First, we explore the features of clustering-based acquisition functions. Figure 6.1 depicts the performance of models trained by different labeling budgets. Here, we only list the results of CodeBERT models and the whole results can be found on our project site.

From the results, we can see that the same acquisition function could train models with significantly different performances depending on the features used. In particular, for the *K-Means*, *K-Center*, and *Coreset* functions, the performance difference is substantial and should not be overlooked. In contrast, the *BADGE* function is relatively stable compared to the others. Then, we analyze the most suitable features that should be used for each function for solving each code task. Table 6.2 displays the percentage of each function that produces models with the best performance across all labeling budgets. Surprisingly, overall, the results demonstrate that the output vectors extracted from the models are superior features compared to code tokens and code embeddings. This suggests that active code learning should focus on output vectors rather than input vectors when conducting data selection, unlike active learning for image and text data. We also use t-test metric to test the signification of the comparison. Table 6.3 summarizes the results. This result also



(a) PC, KM (b) PC, KC (c) PC, BADGE (d) PC, Coreset

(e) CD, KM (f) CD, KC (g) CD, BADGE (h) CD, Coreset

(i) CS-JS, KM (j) CS-JS, KC (k) CS-JS, BADGE (l) CS-JS, Coreset

(m) CS-Ruby, KM (n) CS-Ruby, KC (o) CS-Ruby, BADGE (p) CS-Ruby, Corest

Figure 6.1: Results of different feature-based active code learning. PC, CD, CS, and JS are short for Problem classification, clone detection, code summarization, and JavaScript, respectively.

indicates that the output feature is the best one among our considered features. For example, for clone detection tasks, the output feature beats token and embedding features in KM, BADGE, and Coreset three acquisition functions.

Besides, we observe that the feature selection of some acquisition functions also depends on the types of downstream tasks. In both classification tasks, output vectors and code embeddings are the best choices for *K-Means* and *K-center*, respectively. However, In the non-classification code summarization task, the choices become code tokens and output vectors. On the other hand, output vectors are the most useful features for *BADGE* and *Coreset* regardless of the type of tasks.

Based on the experimental results, we provide recommendations for the feature selection in clustering-based acquisition functions for active code learning.

- **K-Means-C (KM-C)**: use model output vectors (code tokens) for classification (non-classification) tasks.
- **K-Center-C (kC-C)**: use code embeddings (model output vectors) for classification (non-classification) tasks.
- **BADGE-C**: use model output vectors for all code tasks.
- **Coreset-C**: use model output vectors for all code tasks.

Table 6.2: Ratio of trained models achieving best results using different features.

	KM	KC	BADGE	Coreset	Average
Problem Classification					
Token	0%	0%	0%	0%	0%
Embedding	10%	100%	40%	40%	47.5%
Ouput	90%	0%	60%	60%	52.5%
Clone Detection					
Token	0%	40%	0%	0%	10%
Embedding	10%	60%	0%	40%	27.5%
Ouput	90%	0%	100%	60%	62.5%
Code Summarization					
Token	50%	0%	20%	0%	17.5%
Embedding	45%	0%	15%	0%	15%
Ouput	5%	100%	65%	100%	67.5%

Table 6.3: Comparison between different different features (t-test).

	KM	KC	Badge	Coreset
Problem Classification				
Token vs. Embedding	-0.9431	-1.7146	-0.6646	-1.7504
Token vs. Output	-1.6223	-1.6208	-0.6277	-1.6824
Embedding vs. Output	-0.6610	5.2458	-0.2266	0.3993
Clone Detection				
Token vs. Embedding	-1.9726	-0.5992	-1.1039	-2.7805
Token vs. Output	-2.1633	1.8298	-2.3139	-3.1667
Embedding vs. Output	-1.6111	3.7102	-2.6944	-3.0582
Code Summarization				
Token vs. Embedding	-0.2400	2.0836	2.4792	2.8578
Token vs. Output	-7.3257	6.2842	0.5840	6.5609
Embedding vs. Output	-7.0504	4.6275	-1.9706	4.0003

Answer to RQ1: Feature selection highly influences the effectiveness of clustering-based acquisition functions to perform active learning. Surprisingly, in most (62.5%) cases, active learning using output vectors extracted from the models can train a code model with better performance than using code tokens and code embeddings.

6.2.7 RQ2: Acquisition Function Comparison

After studying the feature selection, we compare all acquisition functions on different code tasks. Here, we use our recommended features to build the clustering-based function. Table 6.4 presents the results of classification tasks.

For the multi-class classification task (problem classification), we can see that output-uncertainty-based methods often achieve better results than the clustering-based methods. In which, *Margin* which only uses the top-1 and top-2 probabilities of the outputs performs the best in 5 out of 6 cases. This phenomenon draws a similar conclusion to the previous work [130], that simple methods perform well on active learning. However, considering the binary classification task (clone detection),

Table 6.4: Model performance of active learning trained models for two classification tasks. Values highlighted in red and blue indicate the best and second best. Output: output-based methods. Clustering: clustering-based methods.

		Code Classification						**Clone Detection**					
		CodeBERT			GraphCodeBERT			CodeBERT			GraphCodeBERT		
		1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%
Output	**LC**	29.61	52.90	59.14	33.99	56.93	61.06	80.08	83.51	90.85	79.90	81.73	90.12
Output	**Gini**	43.96	90.95	97.75	44.31	93.02	98.04	85.45	96.58	96.74	84.45	96.41	97.18
-	**Random**	48.01	92.67	95.09	55.05	93.75	95.76	84.22	95.98	96.86	84.83	95.52	96.81
Output	**BALD**	40.63	94.51	97.86	50.83	94.32	98.02	87.61	96.81	97.16	88.73	96.47	97.12
Output	**Entropy**	42.67	90.05	97.67	44.50	92.21	97.99	86.16	96.32	96.89	85.28	96.89	97.17
Output	**Margin**	51.22	95.41	97.91	57.76	96.33	98.16	86.38	96.82	96.92	84.92	97.02	97.17
-	**CAL**	29.99	54.51	64.99	34.04	57.68	65.33	79.57	91.10	95.14	77.60	88.45	92.06
Clustering	**KM**	51.69	94.42	97.65	57.38	95.41	97.99	87.88	97.01	97.10	87.80	96.92	97.23
Clustering	**KC**	47.84	94.01	97.73	56.18	95.57	97.33	87.48	95.07	97.79	85.77	96.85	97.62
Clustering	**Badge**	46.41	94.33	97.39	53.48	95.31	97.91	88.17	97.10	97.27	90.60	97.25	97.09
Clustering	**Coreset**	47.96	94.23	97.66	56.35	95.48	97.24	86.45	96.95	96.96	92.10	97.19	97.06

Table 6.5: Comparison between BADGE and output-based methods (*t*-test). Task: clone detection.

	CodeBERT		
	1%	5%	10%
BADGE-C vs. LC	8.4889	5.9074	6.2645
BADGE-C vs. Gini	2.5009	4.0470	3.1028
BADGE-C vs. Blad	0.4731	1.7192	0.5137
BADGE-C vs. Entropy	2.1334	2.2341	1.7796
	GraphCodeBERT		
BADGE-C vs. LC	23.2590	10.0688	12.3815
BADGE-C vs. Gini	11.4256	3.4707	-0.5947
BADGE-C vs. Blad	1.8989	2.8951	-0.2334
BADGE-C vs. Entropy	10.2222	1.5331	-0.7654

interestingly, the results show that the previous conclusion can not stand. In all cases, clustering-based methods outperform *simple methods* (output-uncertainty-based methods). Table 6.5 shows the results of the comparison between BADGE-C (the best clustering-based method) and all output-based methods on the code clone detection task. The results also demonstrate that except for labeling budget 10%, BADGE-C outperforms *simple methods* in this task. In summary, the first conclusion we can draw is – no acquisition functions consistently perform better than others, and findings [130] from previous works can not be directly applied to active code learning.

Then, move to the non-classification task (two code summarization tasks), table 6.6 and table 6.7 show the results. The first finding is that we will get different conclusions by using different evaluation metrics. For example, the PPL scores demonstrate that *KC-C* is the best acquisition function while the BLEU scores suggest *Coreset-C*. However, regardless of the evaluation metrics we used, the gap between the performance of active learning-trained models and the performance of models trained by using the entire data is big. For example, for JavaScript-CodeBERT, under 10% labeling budget, the best PPL score and BLEU score we get are 5.1313 and 10.09, which are 33.28% and 29.64% lower than the 3.85 and 14.34 computed

Table 6.6: PPL of active learning trained code summarization models with different training budgets. Values highlighted in red indicate the best.

	JavaScript					
	CodeBERT			GraphCodeBERT		
	1%	5%	10%	1%	5%	10%
Random	5.6771	5.3407	5.2038	5.3806	5.1560	5.0938
KM-C	5.6840	5.3586	5.2219	5.3852	5.1589	5.0821
KC-C	5.5978	5.2082	5.1313	5.2883	5.0748	5.0178
BADGE-C	5.6654	5.3214	5.2079	5.3630	5.1379	5.0639
Coreset-C	5.6013	5.2222	5.1334	5.2912	5.0987	5.0346
	Ruby					
	CodeBERT			GraphCodeBERT		
	1%	5%	10%	1%	5%	10%
Random	5.6671	5.4198	5.3662	5.3806	5.1560	5.0938
KM-C	5.6819	5.4662	5.4050	5.4038	5.1843	5.1100
KC-C	5.5565	5.2313	5.1992	5.4366	5.2179	5.1881
BADGE-C	5.6623	5.4067	5.3487	5.5942	5.3490	5.3015
Coreset-C	5.5536	5.2158	5.1796	5.4368	5.2120	5.1412

Table 6.7: BLEU score of active learning trained code summarization models with labeling budget 10%. Values highlighted in red indicate the best.

	JavaScript		Ruby	
	CodeBERT	GraphCodeBERT	CodeBERT	GraphCodeBERT
Random	9.62	10.10	10.36	11.60
KM-C	9.94	10.37	10.15	11.44
KC-C	9.96	10.32	10.46	11.55
BADGE-C	9.41	9.84	10.71	11.70
Coreset-C	10.09	10.42	10.45	11.71

from the model trained by entire training data. These results are totally different from the ones drawn by the classification tasks that using 10% data can train a model with similar and even better performance, e.g., for the clone detection task, models trained with 10% (97.79%) data have better performance than the models trained by entire training data (97.15%). Therefore, we can say that active code learning in non-classification code tasks is still in a very early stage, the conclusions from classification tasks can not be migrated to non-classification tasks.

Answer to RQ2: In contrast to previous work on classification tasks [130], our findings reveal that simple methods are ineffective for the binary code classification task – clone detection. Clustering-based acquisition functions consistently outperform output-uncertainty-based functions in this task. In addition, active learning is ineffective for non-classification tasks such as code summarization, as the performance of models trained via active learning lags behind those trained using the entire dataset by at least 29.64%.

6.3 Exploratory Study

As discussed in Section 6.2, active code learning for non-classification code summarization tasks is still in an early stage. In this section, we tend to explore the

potential directions to propose new effective acquisition functions and mainly focus on non-classification code tasks as existing acquisition functions are effective enough to produce good code models for classification tasks. Since clustering-based acquisition functions are the main techniques used for non-classification tasks, the main goal of our exploratory study is to explore ways to improve this type of acquisition function.

The key idea of clustering-based acquisition functions is to select a diverse set of data, each data sample in this set has big distances from the others. As a result, the calculation of the distance between each data is important in clustering-based functions. Thus, the straightforward way to improve the existing acquisition functions is to provide a precise way to measure the distance between code pairs (or their features). Generally, in the existing acquisition functions, the distance is computed by the Euclidean distance between two vectors that represent two programs. The main concern of this distance calculation is that vectors, e.g., code tokens, code embeddings, and output vectors, highly rely on code representation techniques or code models. It is difficult to say such computed distance can represent the real distance between two programs.

Fortunately, recent research has proposed some evaluation metrics [140, 141] for code generation. Roughly speaking, different from the existing distance methods, these metrics are specifically designed for code to measure the similarity between the machine-generated code snippets and the reference. The studies conducted by the original works show that there is a strong correction between the evaluation metrics and human preference. Inspired by these works, we propose to *use the evaluation metrics as the distance methods* for the clustering-based acquisition functions. Ideally, this new distance should be more precise and the active code learning should be more effective.

6.3.1 Study Design

To validate our conjectures, we empirically explore the following two problems:

- *Is there a correction between the distance of selected data to each other and the performance of trained models based on these data?* Through this study, we explore the probability of improving active code learning from the perspective of providing new distance methods for clustering-based acquisition functions.
- *Is there a connection between the existing distance methods and the code evaluation metrics?* Through this study, we explore whether our proposed distance methods are different from the old ones or not. The positive answer demonstrates that evaluation-metric-based methods cannot be replaced by the existing distance methods.

Addressing the first problem follows the following steps:

Step 1 We prepare two groups of models, the first group contains the initialized models (the same as the initialized models used in section 6.2 – models trained using 500 initial training data) that represent models in the early stage of active learning, and the second group includes models that have already been trained using 5% of training data that represent models at a late stage of active learning. In this way, we can see if the correlation we want to study holds in models with different performances. Note that we have prior knowledge of the data used to train the models.

Step 2 We conduct active code learning by using *Random* acquisition function for each group of models 100 times and record the 100 groups of selected data as

Table 6.8: Correlation between the selected data distance to each other and the performance of trained models based on these data. Each value represents the correlation coefficient. $Model_e$: model at the early stage of active learning. $Model_l$: at the late stage of active learning. Value with * indicates the P-value is less than 0.05.

	Classification (Accuracy)		**Non-classification (PPL)**	
	$Model_e$	$Model_l$	$Model_e$	$Model_l$
Cosine	0.0589	-0.2555*	0.0705	0.1328
Euclidean	-0.0773	-0.2262*	0.0868	0.0947
BLEU	0.0309	-0.0589	0.0504	-0.1699
CodeBERTScore	0.0463	-0.0203	0.0128	-0.1965*

well as the trained models for further analysis.

Step 3 We measure the accuracy of the 100 trained models on the test data and calculate the average distance between the selected data samples in each group. Finally, we can get 100 accuracy values and corresponding 100 distance values. Here, the distance can be calculated by different methods.

Step 4 We use Spearman's rank correlation coefficient to compute the correlation between the accuracy and the distance provided by **step 3**.

For the second problem, we use different distance calculation methods in **Step 3** to obtain the distance scores and then use Spearman's rank correlation coefficient to compute the correlation between these distance methods.

6.3.2 Setup

We select one classification task (problem Classification), and one non-classification task (Code summarization for JavaScript programs) to conduct this exploratory study. For both tasks, we chose CodeBERT as our base model. For the distance calculation methods, we consider four in this study, cosine similarity as distance, Euclidean distance, BLEU score as distance, and CodeBERTScore [141] as distance. Here, CodeBERTScore is a state-of-the-art evaluation metric for code generation. It uses pre-trained contextual embeddings to vectorize each token in the reference program and the generated program first. Then, it computes the pairwise cosine similarity between every embedded token in the reference and every encoded token in the generated code. Finally, the maximum similarity score in each row of the pairwise matrix is used to compute the final similarity of these two programs. Since the BLEU score and CodeBERTScore are computed based on the input level, we also compute the cosine distance and Euclidean distance based on the input embedding (code embedding) for a more fair analysis.

6.3.3 Results Analysis

Table 6.8 presents the results of the correlation between data distance and model accuracy. Surprisingly, the results indicate that regardless of the code tasks, when the model has a poor performance, i.e., at the early stage of active code learning, the trained model performance is not related to the diversity (the distance of data to each other) of used training data. However, for a model that was already trained on a few data and with a good performance, i.e., at the late stage of active code

Table 6.9: Correlation between different distance calculation methods. Each value represents the correlation coefficient. $Model_e$: model at the early stage of active learning. $Model_l$: model at the late stage of active learning. Value with * indicates the P-value is less than 0.05.

	Classification		**Non-classification**	
	$Model_e$	$Model_l$	$Model_e$	$Model_l$
Cosine-Euclidean	0.9438*	0.9155*	0.9717*	0.9769*
Cosine-BLEU	-0.0943	-0.0356	-0.1012	-0.0111
Cosine-CodeBERTScore	-0.2522	-0.0838	-0.4600*	-0.0030
Euclidean-BLEU	-0.0963	-0.0230	-0.0856	-0.0856
Euclidean-CodeBERTScore	-0.2457	-0.0615	-0.4176*	-0.0038
BLEU-CodeBERTScore	0.6464*	0.6464*	0.2422	0.2422

learning, the conclusion changed. Considering the classification task, there is a weak correlation between the cosine and Euclidean distance with the accuracy of the models. That means clustering-based acquisition functions that use these two distance calculation methods are promising to train a model with high accuracy. As already shown in table 6.4, all these acquisition functions based on Euclidean distance achieve good performance in classification tasks. Considering the non-classification task, the results show this correlation becomes weaker, e.g., for cosine similarity, the correlation changes from -0.2555 to 0.1328 in $Model_l$. This is also the reason that the existing acquisition functions do not work well on code summarization tasks and have no advantage over random selection as shown in table 6.6 and table 6.7. However, on the other hand, we can see there is a weak correlation between the evaluation scores-based distance and the accuracy of models which does not happen in our considered two classification tasks. The correlation result of CodeBERTScore on $Model_l$ is significant (with a p-value less than 0.05). These results lead to a promising direction of proposing new acquisition functions that use evaluation metrics as the distance calculation method for the code summarization task.

Takeaway: In non-classification code summarization tasks, our analysis shows that in the selected dataset, greater distances between data samples as calculated by evaluation-metrics-based distance methods lead to better model performance.

Table 6.9 presents the results of the correlation between different distance calculation methods. For models with low performance ($Model_e$), there is always a correlation between cosine similarity or Euclidean distance with CodeBERTScore, which means CodeBERTScore is able to produce similar distance ranking of data to the existing distance methods in these models. Combining the conclusion from the last study, we can see that for $Model_e$, all methods have a similar distance ranking of data, but this ranking is not connected to the performance of models. However, for $Model_l$, we can see there is no correlation between cosine and Euclidean to the BLEU and CodeBERTScore. That is the reason why cosine and euclidean (BLEU and CodeBERTScore) correlate with the model performance while BLEU and CodeBERTScore (cosine and euclidean) do not have this correlation in our considered classification (non-classification) tasks.

Takeaway: Distance methods that are correlated with model performance produce significantly different rankings of data compared to methods that do not exhibit such a correlation.

6.3.4 Case Study

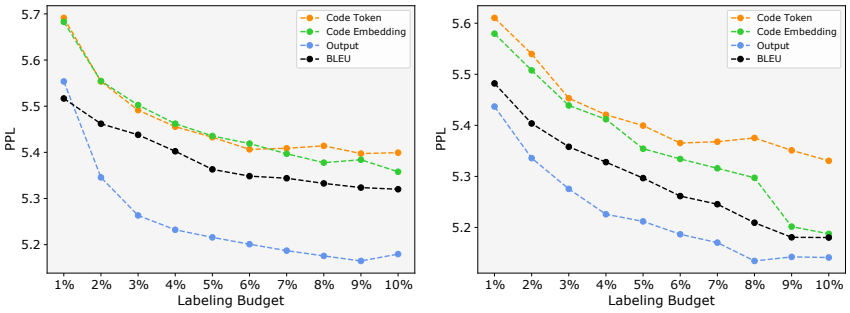
According to our exploratory study, we know that evaluation metrics are promising to be used as distance methods in clustering-based acquisition functions. In this part, we conduct a case study to show if it can really help improve such acquisition functions. Specifically, we modify the distance calculation method in the *Coreset* function from Euclidean distance to BLEU and run experiments on code summarization tasks on Ruby. The reason we chose Ruby is that running evaluation metrics (especially CodeBERTScore) is time-consuming, e.g., it takes more than one month to run an active learning experiment once on code summarization tasks of JavaScript since it contains 2 times more training data than Ruby. The reason for choosing *Coreset* is that it performs the best in code summarization of Ruby as shown in table 6.6 and table 6.7. However, since CodeBERTScore does not support Ruby language now, we can only replace the original distance method in *Coreset* with the BLEU metric. Note that we still use the Euclidean distance between code embeddings to compute *Pairwise Distances* which is the initial step of *Coreset*. It is almost impossible to use the BLEU score here (the time cost is monthly). Algorithm 2 presents the details of the Coreset with BLEU.

Algorithm 2: Coreset with BLEU

Input : X_l: labeled data
X_u: unlabeled data
M: model under training
budget: labeling budget
Output : $X\_seleted$: selected data

$X\_seleted = []$
$Dis = Pairwise\_Distances(X\_u,\ X\_l)$
$init\_data = X\_u(Min(Dis))$
$X\_seleted.append(init\_data)$
$X\_u = X\_u \setminus init\_data$
$count\_num = 1$
while $count\_num < budget$ **do**
  $BLEU\_list = BLEU(init\_data, X\_u)$
  $init\_data = X\_u(Min(BLEU\_list))$
  $X\_seleted.append(init\_data)$
  $X\_u = X\_u \setminus init\_data$
  $count\_num += 1$
end
return $X\_seleted$;

Figure 6.2 depicts the results. We can see that *Coreset* with a BLEU score performs significantly better than *Coreset* with Euclidean distance calculated from code tokens and code embeddings. These results demonstrate the potential of using evaluation metrics as distance methods in active learning. However, using output vectors as clustering features which is also proposed by us is still the best choice which achieves the best results among all the cases. There is a big room to be improved in terms of the performance of trained models and how to propose a new



(a) CodeBERT (b) GraphCodeBERT

Figure 6.2: Active learning with Coreset acquisition functions. Code task: code summarization for Ruby. BLEU: replacing original Euclidean distance in Coreset to BLEU metric.

acquisition function based on the evaluation metrics is still an open problem and our future research.

6.4 Discussion

6.4.1 The Importance of Active Code Learning

Recently, large deep-learning models, especially foundation models like GPT-4 [142] have gained huge attention and achieved many state-of-the-art results in various application domains. This hot trend almost changes the research focus of ML4Code from designing new code model architectures or code representation techniques to how to reuse these foundation models for our specific code tasks. Generally, model reuse involves a fine-tuning step that further optimizes the model parameters and improves performance. As a result, active code learning becomes more and more important since it allows us to fine-tune the pre-trained models with a controllable human effort, i.e., budgets allocated for labeling the datasets used in fine-tuning. This technique provides opportunities for researchers and developers with limited resources to leverage and improve existing big models.

6.4.2 Threat to Validity

The **external threat** lies in our considered acquisition functions used for active learning, code tasks and datasets, and code models. For the acquisition functions, we collect 10 functions that are specifically proposed for active learning and already studied the most in recent works [8, 130]. Other functions such as neural coverage methods are not considered since they are proposed for a different purpose. For code tasks and datasets, we consider both classification tasks that study important problems, problem classification, clone detection, and code summarization. For code models, we prepare two well-known code pre-trained models. Based on our open-source projects, other tasks, and models can be easily added to our benchmark. The **internal threat** can be the implementation of acquisition functions and code models. All implementations of acquisition functions are based on the existing active learning works [8, 143, 144] and after carefully checking. The implementation of code models is also modified from the famous open source project [135]. The **construct threat** can be the configuration of active learning. Since this is the first work that

studies active code learning, we follow our best practice to initialize the code models and set the labeling budgets. Besides, since we compare code models under the same labeling budgets, the comparison results are not affected by the configuration of active learning.

6.5 Conclusion

This work introduced the first benchmark and an empirical study for the important yet unexplored problem – active code learning. Our experimental results demonstrated that active code learning is effective in training code models with expected high performance for classification tasks such as problem classification and clone detection. However, it is still in the early stage for non-classification tasks like code summarization. Besides, we conducted an exploratory study to show using evaluation metrics as distance calculation methods is a promising way to propose new clustering-based acquisition methods. We believe that our benchmark as well as empirical studies will provide developers and researchers insights into efficiently reusing (i.e., with little human effort) existing large pre-trained models for their specific code tasks.

7 *Aries*: Efficient Testing of Deep Neural Networks via Labeling-Free Accuracy Estimation

In this chapter, we propose a novel label-free method, Aries, to estimate the performance of DNN models on unlabeled test sets. Our method capitalizes on the underlying insight that a correlation often exists between the accuracy of data predictions and their proximity to the decision boundary. More specifically, given two datasets whose distribution of the distance to the decision boundary is close, they could share a similar prediction accuracy.

Contents

7.1 Introduction

Deep learning (DL) has been continuously deployed and applied in different industrial domains that impact our social society and daily life, such as face recognition [145, 146], autonomous driving [147, 148], and video gaming [149, 150]. As the core of DL-enabled systems, deep neural network (DNN) follows the data-driven development paradigm and learns the decision logic automatically based on the incorporated learned knowledge of training data. Similar to traditional software that needs to be well tested, DNNs also need to be comprehensively evaluated before deployment to reduce potential risks in the real world [18, 19].

A common *de facto* standard to assess the quality of DNNs in the industry is by evaluating DNNs on a collected set of labeled data. In practice, when building a DNN model, developers often split a dataset into the training set, validation set, and test set. The test set is mainly used to measure the accuracy of the trained model (as an indicator of performance generality), thus, the final developed DNN often comes with the reported test accuracy. However, the original test set often only covers a part of the data distribution (generally, the same distribution as the training data). The distribution of unseen data is often unclear in the practical context, and the reported test accuracy is hard to reflect the actual model performance in real usage. Therefore, in addition to testing models on the original test data, it is highly desirable to conduct a performance evaluation of DNNs on new data, which is generally available from a large amount of daily or monthly incoming data.

However, different from the original test data that already have been labeled, the unseen/new data are usually raw with the absence of label information. It is challenging to assess a model on unlabeled data. More importantly, labeling all the new data (that could be large in size) is labor-intensive and time-consuming, which is almost impossible and impractical. For some complex tasks, domain knowledge from experts is mandatory, leading to the labeling even harder. For example, it takes more than 600 man-hours for experienced software developers to label all the codes from 4 libraries [3].

Towards addressing the data labeling issue for more efficient DNN testing, recently, researchers have recently adapted the test selection concept [151, 152] from the software engineering community to select and label a subset of representative data, then test and assess the model accordingly. For example, Li *et al.* proposed cross entropy-based sampling (CES) [6] to select a subset that has the minimum cross-entropy with the entire test dataset to test the DNN. In this way, the labeling effort can be significantly reduced and the testing has acceptable bias. However, although test selection is a promising direction for efficient DNN testing, the labeling efforts and costs persist. To bridge the gap, in this paper, we aim to automatically estimate the test accuracy without extra manual labeling.

To this end, we propose a novel technique, *Aries*, to efficiently estimate the performance of DNNs on the new unseen data based on existing labeled test data. The intuition behind our technique is: **there can be a connection between the prediction accuracy of the data and the distance of the data from the decision boundary.** More specifically, given two datasets whose distribution of the distance to the decision boundary is close, they could share a similar prediction accuracy. A preliminary empirical study is first conducted to validate our assumption. Specifically, we adopt the existing dropout-based uncertainty to estimate the distance

between the data instances and the decision boundary. By splitting the uncertainty score into n intervals, we obtain n buckets of the distance distribution. Then, we can map the data instances into a bucket based on their uncertainty scores. With this, we can estimate the accuracy of each bucket based on the original test data (with labels) that fall into this bucket as the supporting evidence (points). Finally, given the new data without labels, we map them into different buckets and leverage the estimated bucket accuracy to calculate the overall accuracy of the new data. Compared to existing techniques that need the labeled data to calculate the model accuracy, *Aries* is fully automatic without extra human labeling effort.

To assess the effectiveness of *Aries*, we conduct in-depth evaluations on two commonly used datasets, CIFAR-10 and Tiny-ImageNet, four different DNN architectures, including ResNet101 and DenseNet121. Besides the original test data, we also use 13 types of transformed test sets (e.g., data with added brightness) to simulate the new unlabeled data that could occur in the wild, where the transformed data could follow different data distributions from the original test data [153]. The results demonstrated that *Aries* could precisely estimate the performance of DNN models on new unlabeled data without further labeling effort. Compared to the real accuracy, the estimated accuracy exhibits a difference ranging from 0.03% to 2.60% by using the default parameter setting. Besides, compared to the state-of-the-art (SOTA) labeling-free model performance estimation methods [5], *Aries* can predict more accurate accuracy. And compared to the test selection methods CES [6] and PACE [7], which need to label a portion of test data, *Aries* can still achieve competitive results without extra labeling. Moreover, we conduct ablation studies to explore the impact of each component of *Aries* on the estimation performance.

To summarize, the main contributions of this paper are:

- We propose a novel DNN testing technique for quality assessment, *Aries*, that can efficiently estimate the model accuracy on unlabeled data without any labeling effort.
- We empirically demonstrate that *Aries* can achieve better results than SOTA labeling-free model performance estimation methods and competitive results compared to test selection methods that require human labeling effort.
- We also comprehensively explore each potential factor that could affect the performance of *Aries* and provide the recommendation parameters.
- We release all our source code publicly available[1], hoping to facilitate further research in this direction.

7.2 Methodology

We first introduce the insight and assumption of *Aries*, then conduct preliminary studies to empirically validate our assumptions, and finally present the details of *Aries*.

7.2.1 Key Insight and Assumption

Our assumption is that there could be a correlation between the prediction accuracy and the distance of the data from the decision boundary. To better understand this assumption, Fig. 7.1 depicts an intuitive example of a binary classification with a decision boundary (blue solid lines) splitting the data space into 2 regions. Since

[1]https://github.com/wellido/Aries

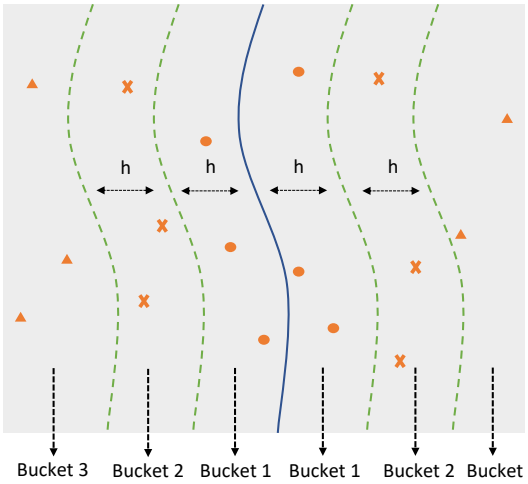


Figure 7.1: An example of the assumption of our technique. Data in the same bucket are highlighted with the same marker.

the data falling into the same region will be predicted by the same label, but with different confidences, we split each region further into multiple sub-regions based on its distance to the boundary. In the figure, we evenly split the space into 3 buckets (*Bucket1*, *Bucket2*, and *Bucket3*) by the distance h. We assume that, for data in the same bucket, the probability of making the (in)correct prediction is similar, i.e., the model has the same performance on these data. Thus, if we can obtain the model accuracy in each bucket and map new data into the corresponding bucket, we can approximate the model accuracy on these new data.

The essential insight of our assumption is to measure the distance between data and decision boundaries (i.e., how to define h in Fig. 7.1). Given the fact that the data space is usually high dimensional and complex, it is difficult to directly describe the decision boundary. Recently, the dropout uncertainty [154, 155] has been widely used to estimate the distance of the data from the decision boundary. Roughly speaking, given one data and a model with dropout layers. After using this model to predict the data multiple times, if the predicted outputs have a huge variance, we say this data might be uncertain by the model and near the decision boundaries. We employ the dropout uncertainty in *Aries* for the distance approximation and will explore other options in the configuration study. At a high level, our insight is sound and practical in the way that the prediction confidence of a DNN follows a particular distribution that is automatically learned from the training data in the training process. When a test sample falls into a distribution and prediction confidence region, our fine-grained split region could provide a certain level of evidence to support DNN quality assessment.

7.2.2 Preliminary Study

First, we define *Label Variation Ratio*(LVR) to approximate the distance between data and decision boundaries.

Definition 1 (**Label Variation Ratio (LVR)**)**.** Given a model M with dropout layers and an input data x, the number of dropout predictions T, LVR of x is defined as:

$$LVR\,(M, x, T) = \frac{\left|\left\{k \mid 1 \leq k \leq T \wedge L_{M^k(x)} = L_{max}\right\}\right|}{T}$$

where $L_{M^k(x)}$ is the k-th predicted label of x by M and L_{max} is the dominant label of T predictions (i.e., the label predicted by most predictions). Intuitively, the prediction of the data near the boundary has low confidence, i.e., with lower LVR.

We then divide the data space into n buckets by splitting the LVR into n equal intervals, where the range of LVR is (0,1]. For example, if n is 2, then we have two buckets, and the corresponding LVR intervals are (0, 0.5] and (0.5, 1]. A data instance falls into a region based on its LVR value.

Definition 2 (**Bucket**)**.** Given a dataset X, M and the number of intervals n, we define the data that belong to t^{th} bucket as:

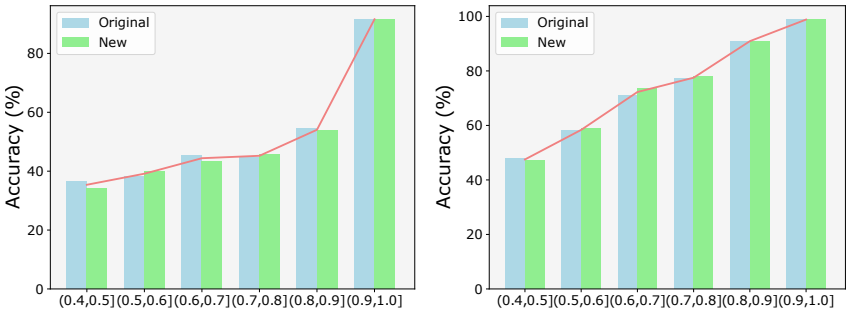
$$Bucket(X, t, T) = \left\{x \mid x \in X \wedge \frac{t}{n} < LVR(M, x, T) \leq \frac{t+1}{n}\right\}$$

where $0 \leq t < n$ and T is the number of dropout predictions.

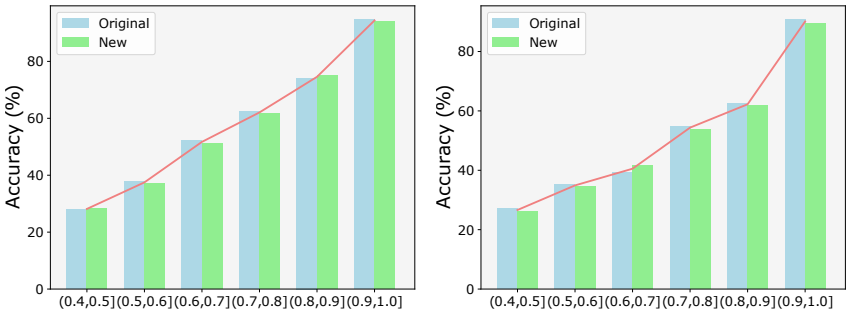
Next, to validate the rationality of our assumption, we conduct two preliminary studies to check 1) if data in the same *Bucket* have similar accuracy, and 2) if there is a relation between LVR and model accuracy (please refer to Section 7.3 for details of datasets and models). In the first study. We randomly split the test data into two sets and assume one set we already have the label information and the other is the new unlabeled data. Then we activate the Dropout layers in each model to predict these two sets multiple times (here, we set the number of T as 50, which is the default setting of *Aries*). Afterward, we calculate the LVR score of each data and then split the data into different *Buckets* using Definition 2. Finally, we check the accuracy of the model on the data that are in the same *Bucket*. Fig. 7.2 presents the results of this study. Note that we eliminate the results of data whose LVR scores are smaller than 0.4 due to the negligible amount (e.g., only one in CIFAR-10, ResNet20). We can see that 1) the two sets of data have similar accuracy in the *Buckets* regardless of the datasets and models, 2) data with higher LVR scores (closer to the decision boundary) have higher accuracy. This finding justifies our assumption.

Finding 1: A DNN has similar accuracy on the data sets that have similar distances to the decision boundary.

Second, we investigate if there is a relation between the size of data with high LVR and the accuracy of the model on this set. Usually, data with a high LVR score means the model is confident in predicting this data. Intuitively, the size of data that the model has high confidence could partly reflect the model's performance. Fig. 7.3 depicts the model accuracy (x-axis) and the percentage of highly confident data (y-axis) on each data set. Here, the highly confident data means their LVR is 1. We can see that there is a clear linear relationship between the two values. The results lead to our basic idea: we can measure the percentage of highly confident data in the new unlabeled data although we do not know the truth labels of the new data. Then, based on existing test data with truth labels, we can measure the accuracy of the datasets with a certain ratio of highly confident data. Finally, we can estimate the accuracy of unlabeled data.

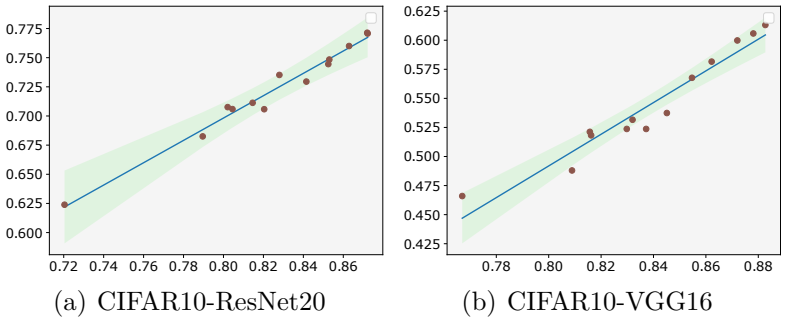


(a) CIFAR10-ResNet20 (b) CIFAR10-VGG16

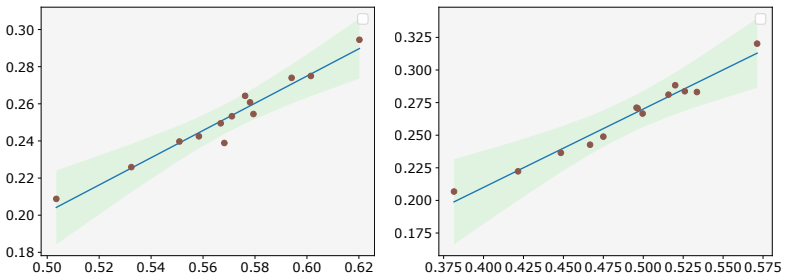


(c) ImageNet-ResNet101 (d) ImageNet-DenseNet121

Figure 7.2: Accuracy in different *Buckets*. Original: labeled test data, New: unlabeled new unlabeled data.



(a) CIFAR10-ResNet20 (b) CIFAR10-VGG16



(c) ImageNet-ResNet101 (d) ImageNet-DenseNet121

Figure 7.3: The linear relation (blue line) between the size of data with the highest label variation ratio (y-axis, unit: ratio) and the test accuracy (x-axis, unit: %). We apply the least squares polynomial fit to draw the blue line in each figure.

Finding 2: There is a linear relationship between the % of highly confident data ($LVR = 1$) and the accuracy of the whole set. Therefore, given some labeled data, if we know 1) the accuracy of the DNN in each *Bucket*, and 2) the percentage of highly confident data, it is promising to estimate the accuracy of the new unlabeled data.

7.2.3 *Aries*: Efficient Testing of DNNs

Algorithm 3: *Aries*: efficient testing of DNNs

Input : M: model with dropout layer
X_{ori}: original test data with labels
X_{new}: new data without labels
T: number of dropout predictions
n: number of buckets

Output : Acc_{Aries}: estimated accuracy of X_{new}

1 $correct\_num = 0$
for $i = 0 \rightarrow n-1$ **do**
2 $\quad BucketAcc_i = evaluate(M, Bucket(X_{ori}, i, T))$;
3 $\quad correct\_num+ = |Bucket(X_{new}, i, T)| \times BucketAcc_i$;
end
4 $Acc_{bucket} = correct\_num/len\,(X_{new})$;
5 $Acc_{ori} = evaluate(M, X_{ori})$;
6 $Acc_{confident} = \frac{|Bucket(X_{new}, n-1)|/len(X_{new})}{|Bucket(X_{ori}, n-1)|/len(X_{ori})} \times Acc_{ori}$;
7 $Acc_{Aries} = average\,(Acc_{bucket},\ Acc_{confident})$;
return Acc_{Aries};

Based on the two findings, we propose a novel technique, *Aries*, that can test the performance of DNN models without requiring the label information. The key idea of *Aries* is to take advantage of the labeled test sets to approximate the model performance on the new unlabeled data. Algorithm 3 presents the details of our technique which contains two main steps.

First, given a model M with dropout layers, the original labeled data X_{ori}, the number of buckets n, and the dropout prediction time T, *Aries* splits X_{ori} into n buckets according to Definition 2, and calculate the accuracy of the data in each bucket $BucketAcc_i$ (lines 2 and 3). Then, the same as the X_{ori}, we split X_{new} into n buckets and use the bucket accuracy $BucketAcc_i$ to estimate the correctly predicted number $correct\_num$ of X_{new} (line 4).

Then, we perform the accuracy estimation. First, according to finding 1, *Aries* directly computes the accuracy using the correct data number of new data in each bucket and produces the first estimation Acc_{bucket} (line 6). Then, based on finding 2, we calculate the accuracy of the labeled data first Acc_{ori} (line 7). Afterward, *Aries* computes the proportion of the high confident data in the new unlabeled data to the original test data, and then estimates the second accuracy $Acc_{confident}$ (line 8). In the end, since in practice, the Acc_{bucket} ($Acc_{confident}$) over(under)-estimates the accuracy, we compute the average of the two estimated accuracy as the final output of *Aries*, Acc_{Aries}(line 9). We will detail explain why we combine Acc_{bucket} and $Acc_{confident}$ in section 7.4. Highlight that, *Aries* only requires N forward propagations to work, which is significantly more efficient than the existing training-based methods [5].

Example: Taking the cases in Fig. 7.1 as an example with 3 Buckets ($n = 3$), $Bucket1$, $Bucket2$, and $Bucket3$. We assume that the accuracy of original test data (Acc_{map}) in $Bucket1$, $Bucket2$, and $Bucket3$ are 60%, 70%, and 80%, respectively. And the number of original test data ($BucketSize_{ori}$) in each part is 200, 300, and 400, respectively. Then, for the new unlabeled data, the number of data in each section ($BucketSize_{new}$) is 300, 400, and 500. Then the Acc_1 is calculated

Table 7.1: Details of datasets and DNNs

Dataset	Classes	Training	Test	DNN	Parameters	Accuracy (%)
CIFAR-10	10	50k	10k	ResNet20	274442	87.44
				VGG16	2859338	91.39
Tiny-ImageNet	200	100k	10k	ResNet101	43036360	74.09
				DenseNet121	7242504	70.70

by $(200 \times 60\% + 300 \times 70\% + 400 \times 80\%)/(200 + 300 + 400) = 72.22\%$. Next, assume that the accuracy of the original test data is 70%, the Acc_2 is calculated by $((500/1200)/(400/900)) \times 70\% = 66.35\%$. The final output of *Aries* is $(72.22\% + 66.35\%)/2 = 69.29\%$.

In *Aries*, the dropout prediction plays an important role in estimating the distance to decision boundaries. Therefore, the dropout rate is the first potential influencing factor. Next, the number of buckets that determines the splitting granularity could be the second influencing factor. In addition, since our technique uses dropout prediction and label change times to approximate the distance between the data and the decision boundaries, the distance approximation method is the third influencing factor. In Section 7.4.2 we will explore the influence and the best settings of *Aries* for these three factors.

7.3 Experimental Setup

To evaluate the effectiveness of *Aries*, we conduct experiments on two popular datasets, four DNN architectures, and 13 types of data transformations that are utilized to generate new unlabeled data. Our in-depth evaluation answers the following research questions:

RQ1: How effective is *Aries* in accuracy estimation?

RQ2: How does each component affect and contribute to the effectiveness of *Aries*?

Subject datasets and DNN models. Table 7.1 presents the details of datasets and models. CIFAR-10 [156] is a 10-class dataset of color images, e.g., airplanes and birds. For this dataset, we build two models, ResNet20 [157] and VGG16 [109]. Tiny-ImageNet [158] is a more complex dataset that contains 200 image categories, e.g., goldfish and monarch. For this dataset, we use ResNet101 and DenseNet121. In our experiments, we take the original test data for the decision boundary analysis to estimate the accuracy of new data.

For the new unlabeled data preparation, we directly use the popular natural robustness benchmark dataset [159] for our experiments. This benchmark provides two datasets, CIFAR-10-C and Tiny-ImageNet-C, that are generated by adding common corruptions and perturbations into the original test data, e.g., by increasing the brightness of the image. It is widely used for evaluating the model performance on distribution-shifted data (unseen data). Besides, a recent study [153] also demonstrates that these kinds of corrupted data can be regarded as out-of-distribution data, because the distance between these data and the original test data is farther than the distance between the real out-of-distribution data and the original test data. In our evaluation, we collect 13 common types that CIFAR-10 and Tiny-ImageNet both include, shown in Table 7.2.

Table 7.2: Details of data transformation methods used for generating new unlabeled data.

Data Type	Description
Brightness	Increase the brightness of the image data
Contrast	Increase the contrast of the object
Defocus Blur (DB)	Add the defocus blur effect to the image
Elastic Transform (ET)	Elastic deformation of images
Fog	Add fog effect to the image
Frost	Add frost effect to the image
Gaussian Noise (GN)	Add Gaussian Noise perturbation to the image
Jpeg Compression (JC)	Change the image to Jpeg format
Motion Blur (MB)	Add the motion blur effect to the image
Pixelate	Convert image to pixelate style
Shot Noise (SN)	Add noise by using the Poisson process
Snow	Add snow effect
Zoom Blur (ZB)	Zoom the image data

Baseline. We first compare *Aries* with two SOTA labeling-free model performance estimation methods:

1. **Predicted Score-based Method (Predicted Score)** assumes that a test sample is correctly classified when its maximum output probability is greater than a threshold τ. In the experiments, we follow the same setting as [5] to set the τ as 0.7, 0.8, and 0.9.

2. **Meta-set** [5] is recently proposed three-step method. First, Meta-set generates multiple diverse test sets by performing different image transformations on the original test set. Then, it computes the Frechet Distance (FD) between the internal outputs of generated and original test sets. Finally, a regression model is trained by using the FD score and the accuracy of test sets. When new data come, Meta-set calculates its FD score with the original test set and then utilizes the regression model to estimate its accuracy. In the experiments, we utilize both linear regression and neural network regression to build the model. The sizes of the meta set and sample set are set as 1000 and 10000, respectively, following the original paper. In addition, we considered the number of image transformations as 1, 2, and 3, whereas the original paper only studied 3. In this way, we build a strong baseline.

Then, we compare *Aries* with existing test selection-based model evaluation methods, Cross Entropy-based Sampling (CES) [6] and Practical Accuracy Estimation (PACE) [7], as baselines. Besides, we also consider random selection as the third baseline. Remarkably, all these three baselines require selecting and labeling a subset from the new unlabeled data to perform testing. We follow the same configuration as the paper [7] to set the labeling budget from 50 to 180 in intervals of 10 for test selection methods.

1. **Cross Entropy-based Sampling (CES)** selects the data that have the minimum

Table 7.3: Details of our used model-level mutation operators.

Level	Operator	Description
Weight	Gaussian fuzzing	Fuzz the weights using Gaussian noise
	Weight Shuffle	Shuffle the weights in the same neuron
Neuron	Neuron Effect Block	Block a neuron effect, i.e., change the output to 0
	Neuron Activation Inverse	Invert the activation status
	Neuron Switch	Switch two neurons in the same layer

cross-entropy with the entire test dataset. It starts from a randomly selected small size of data and iteratively increases the size by adding other data while controlling the cross-entropy.

2. **Practical Accuracy Estimation (PACE)** first clusters data based on the output of the last hidden layer into different groups by using the hierarchical density-based spatial clustering of applications with noise clustering method, then utilizes the example-based explanation algorithm MMD-critic to select the most representative data from each group to label and test the model.

***Aries* configuration.** To reduce the number of hyperparameters in *Aries*, we set the dropout prediction number to be the same as the number of *Buckets* ($T = n$ in algorithm 3), which means in each *Bucket*, all the data have the same LVR score. Then, there are two remaining hyperparameters we need to set 1) the number of *Buckets* we split and 2) the dropout rate. The default setting of the number of *Buckets* and the dropout rate in RQ1 is 50 and 0.5, respectively. In RQ2, we study the different *Bucket* number settings 10, 50, 100, and 150, and different dropout rate settings 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9. Since the setting space is infinite and it's impossible to study all, we recommend the best one among our studied settings and show that by this setting we can already get acceptable estimation results.

Model mutation. In RQ2, we investigate if the DNN model mutation [160, 161] can be used to replace the dropout prediction for the distance of data to boundary approximation. Similar to the dropout, mutation can produce a variant of the original model with a similar accuracy without retraining the model from scratch [160]. The difference is that the dropout technique tends to fully ignore some randomly selected neurons, while the mutation technique can also work on the weight of neurons and the neuron status. First, we randomly use the weight-level and neuron-level mutation operators provided by [160] to generate mutants. The detailed information of the operators is presented in table 7.3. To preserve the quality of the mutants, we set the mutation ratio as 0.1 (0.01) for CIFAR-10 (Tiny-ImageNet) models, and the accuracy threshold as 0.9. Then, we follow the steps in Algorithm 3 to estimate the accuracy of the model on the new unlabeled data.

Implementation and environments. We implement *Aries* in Python based on TensorFlow [59] framework. For the baselines CES and PACE, we utilize their original implementation. For model mutation, we use the available mutation framework provided by the authors. We conduct all the experiments on a 2.6 GHz Intel Xeon Gold 6132 CPU with an NVIDIA Tesla V100 16G SXM2 GPU. To counteract randomized factors, we repeat all the experiments 5 times and report the average

results in this paper. Due to the page limit, we put more detailed results as well as the source code for reproducible study of this paper at the companion site[1].

7.4 Results Analysis

7.4.1 RQ1: Effectiveness of *Aries*

Effectiveness on in-distribution data. First, we evaluate the accuracy estimation effectiveness of the data (the two sets randomly split from the original test data) used in our preliminary study (Section 7.2.2). Table 7.4 lists the results. Recall that Acc_{bucket} is the accuracy estimated by the *Bucket* accuracy. $Acc_{confident}$ is the accuracy estimated by using the size of high confidence data. Acc_{Aries} is the final estimated accuracy of *Aries*, the weighted sum of Acc_{bucket} and $Acc_{confident}$. We report Acc_{bucket} and $Acc_{confident}$ to verify the importance of each component of our technique. The results demonstrate that all three estimations can predict the model accuracy on the new data that follow the same data distribution as the original test data with the difference slightly ranging from 0.07% to 1.02%. In addition, it's hard to determine which estimation is the best since one can perform better or worse than the others in different datasets and models.

Table 7.4: Estimated accuracy and the absolute difference between estimated accuracy and real accuracy(%) on the test data (New) used in the preliminary study. *Real*: real accuracy. Acc_{bucket}, $Acc_{confident}$, and Acc_{Aries} refer to lines 6, 8, and 9 in Algorithm 3, respectively. The best is highlighted.

	CIFAR-10		**Tiny-ImageNet**	
	ResNet20	**VGG16**	**ResNet101**	**DenseNet121**
Real	87.28	91.18	74.16	71.98
Acc_{bucket}	87.41 (0.13)	91.33 (0.15)	73.73 (0.43)	71.93 (0.05)
$Acc_{confident}$	87.35 (0.07)	90.81 (0.37)	73.39 (0.77)	70.96 (1.02)
Acc_{Aries}	87.38 (0.10)	91.07 (0.11)	73.56 (0.60)	71.45 (0.54)

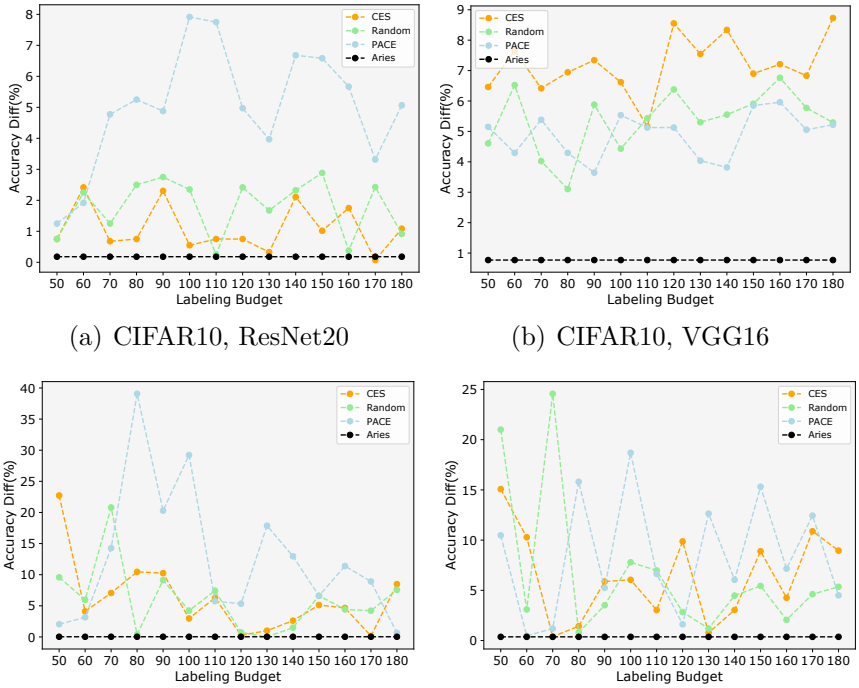
Effectiveness on data with distribution shift. In real-world applications, software developers are more interested in the data that follow various data distributions since after the model has been deployed in the wild, the distribution of new unlabeled data is uncontrollable. Thus, we evaluate *Aries* using the data that contain different data distributions. Table 7.5 summarizes the results of the accuracy estimation on the 13 types of distribution-shifted test sets. The same to the results of the preliminary study, we report all three estimations here to analyze the contribution of each part. Overall, different from the results on the original test data, the Acc_{Aries} is closer to the real accuracy in most cases (42 out of 52) than Acc_{bucket} and $Acc_{confident}$. On average, compared to the real accuracy, the difference of estimated accuracy Acc_{Aries} ranges from 0.03% to 2.60% across all the datasets and models. And for each models, the average difference of Acc_{Aries} is smaller than 1% (0.52%, 0.91%, 0.27%, and 0.85% for ResNet20, VGG16, ResNet101, and DenseNet121). Surprisingly, for Tiny-ImageNet, ResNet101, which has the most complex model architecture among all the models we considered, all the estimated biases are smaller than 0.59%. This means, our technique is flexible and still effective on challenging datasets and models.

More specifically, the results reveal that, when we utilize *Aries* to estimate the accuracy of distribution shifted data, only considering the Acc_{bucket} or $Acc_{confident}$

Table 7.5: Difference between estimated and real accuracy (%) under data distribution shifts. *Real*: real accuracy,Acc_{bucket}, $Acc_{confident}$, and Acc_{Aries} refer to lines 6, 8, and 9 in Algorithm 3, respectively. Meta-set-Linear(NN)-X means Meta-set with linear (neural network) regression and X types of image transformation combination. The best estimation is highlighted.

Dataset	DNN		Brightness	Contrast	DB	ET	Fog	Frost	GN	JC	MB	Pixelate	SN	Snow	ZB	Avg.
CIFAR-10	ResNet20	Real	87.07	85.25	87.22	79.11	86.29	82.04	87.2	81.46	78.21	82.8	78.96	80.45	76.4	82.50
		Predicted score ($\tau = 0.7$)	6.48	6.72	5.81	9.78	6.22	8.59	5.82	5.58	10.76	8.92	10.51	10.49	10.09	8.14
		Predicted score ($\tau = 0.8$)	3.46	3.11	2.61	4.87	2.96	4.35	2.65	0.43	5.56	5.18	5.75	6.03	4.74	3.98
		Predicted score ($\tau = 0.9$)	1.33	2.2	1.95	2.38	1.79	1.48	1.99	8.27	1.98	0.18	0.97	0.15	3.37	2.16
		Meta-set-Linear-1	8.91	7.86	9.16	2.76	8.41	5.38	9.14	4.56	3.48	5.65	3.75	3.69	3.00	5.83
		Meta-set-Linear-2	25.87	24.57	26.09	19.13	25.28	21.85	26.07	21.11	19.33	22.28	19.75	20.20	18.41	22.30
		Meta-set-Linear-3	25.69	24.41	25.91	19.01	25.10	21.72	25.89	20.97	19.26	22.13	19.67	20.06	18.39	22.17
		Meta-set-NN-1	5.11	7.99	4.51	4.85	8.16	10.87	4.53	9.21	3.30	8.27	11.36	6.52	2.87	6.73
		Meta-set-NN-2	7.85	11.76	8.24	11.24	11.61	12.77	8.23	10.56	11.11	6.23	10.44	8.94	11.50	10.04
		Meta-set-NN-3	3.71	2.85	3.88	5.53	1.41	7.44	3.91	3.96	4.90	3.55	5.34	4.97	7.86	4.56
		Acc_{bucket}	0.26	0.90	0.09	4.03	0.42	2.73	0.10	2.31	4.33	2.91	3.75	3.24	5.04	2.32
		$Acc_{confident}$	0.23	1.25	0.59	2.58	0.83	1.89	0.60	1.20	3.86	0.16	2.27	1.36	4.97	1.68
		Acc_{Aries}	0.25	0.18	0.34	0.72	0.21	0.42	0.35	0.55	0.23	1.54	0.74	0.94	0.04	0.50
	VGG16	Real	91.41	90.96	91.78	88.69	91.15	87.39	91.78	87.12	89.55	88.81	85.97	85.77	88.55	89.15
		Predicted score ($\tau = 0.7$)	4.93	4.96	4.58	6.04	4.83	7.39	4.54	7.73	6.12	6.24	7.81	8.13	6.15	6.11
		Predicted score ($\tau = 0.8$)	2.65	2.57	2.62	3.41	2.59	4.61	2.58	5.24	3.82	3.85	5.33	4.89	3.56	3.67
		Predicted score ($\tau = 0.9$)	0.78	1.75	0.90	0.35	0.69	2.75	0.85	1.05	1.44	0.42	1.11	1.95	1.34	1.18
		Meta-set-Linear-1	6.16	6.68	6.59	5.46	6.23	3.75	6.58	2.51	7.25	3.99	2.10	1.31	8.30	5.15
		Meta-set-Linear-2	11.66	12.10	12.08	10.79	11.70	9.11	12.07	7.95	12.50	9.45	7.48	6.74	13.37	10.54
		Meta-set-Linear-3	17.41	17.74	17.83	16.30	17.41	14.67	17.82	13.63	17.90	15.16	13.07	12.40	18.52	16.14
		Meta-set-NN-1	162.32	150.67	157.85	135.52	153.88	148.12	157.93	141.46	137.33	154.80	145.26	146.55	119.96	147.05
		Meta-set-NN-2	18.56	9.53	16.85	6.00	12.95	9.27	16.79	10.99	5.89	15.56	8.86	13.11	6.09	11.57
		Meta-set-NN-3	16.22	27.11	15.89	27.55	22.52	29.10	15.98	24.48	29.34	21.22	29.72	26.21	33.94	24.56
		Acc_{bucket}	0.01	0.25	0.01	0.82	0.19	1.64	0.00	1.93	0.94	1.05	2.33	2.31	0.82	0.95
		$Acc_{confident}$	0.65	1.79	1.47	0.99	0.51	6.85	1.26	3.38	1.99	5.68	1.78	5.21	2.45	2.62
		Acc_{Aries}	0.33	0.77	0.74	0.08	0.16	2.60	0.63	0.73	0.53	2.19	0.27	1.45	0.81	0.87
Tiny-ImageNet	ResNet101	Real	63.93	50.18	55.77	55.33	60.23	57.6	57.96	59.58	56.86	62.37	57.17	57.81	53.15	57.53
		Predicted score ($\tau = 0.7$)	53.92	47.55	50.88	49.74	52.42	50.94	50.08	55.57	50.07	54.83	51.31	47.76	45.53	50.82
		Predicted score ($\tau = 0.8$)	60.77	51.87	55.05	54.09	57.68	56.46	56.27	57.80	55.42	59.58	55.63	55.59	53.37	56.12
		Predicted score ($\tau = 0.9$)	62.99	50.10	54.61	53.81	58.69	56.75	57.02	58.82	55.56	61.52	56.23	55.85	52.52	56.50
		Meta-set-Linear-1	2.59	7.31	2.00	3.47	0.76	3.32	1.3	1.32	1.61	0.75	1.11	2.82	3.36	2.44
		Meta-set-Linear-2	19.38	5.55	11.15	10.73	15.68	13.04	13.37	15.02	12.25	17.83	12.56	13.25	8.5	12.95
		Meta-set-Linear-3	28.84	14.99	20.58	20.17	25.13	22.5	22.81	24.48	21.69	27.29	22	22.7	17.93	22.39
		Meta-set-NN-1	4.9	11.4	2.73	4.06	0.28	2.57	3.75	3.46	1.92	3.28	3.56	1.92	5.79	3.82
		Meta-set-NN-2	14.37	3.48	7.75	6.46	12.57	13.03	14.25	13.78	8.65	13.97	13.91	11.91	4.66	10.68
		Meta-set-NN-3	25.6	16.54	18.91	18.09	24.32	25.21	26.73	24.38	19.76	25.03	26.52	23.48	16.43	22.38
		Acc_{bucket}	4.48	10.76	7.80	7.61	5.70	7.00	7.03	6.26	7.27	5.22	7.27	6.55	9.16	7.09
		$Acc_{confident}$	5.06	10.83	7.03	7.45	6.81	6.44	6.58	5.97	7.12	5.32	6.66	7.73	10.25	7.17
		Acc_{Aries}	0.29	0.03	0.38	0.08	0.55	0.28	0.23	0.15	0.08	0.05	0.31	0.59	0.54	0.27
	DenseNet121	Real	58.26	36.19	44.9	46.94	53.58	49.83	51.85	52.37	47.54	57.4	51.26	49.74	42.28	49.40
		Predicted score ($\tau = 0.7$)	5.67	7.86	12.39	10.69	7.41	9.92	6.82	8.32	11.13	5.75	7.43	10.51	10.94	8.83
		Predicted score ($\tau = 0.8$)	15.00	4.98	10.42	12.74	13.68	12.53	17.20	13.42	11.92	15.06	14.78	12.78	9.79	12.64
		Predicted score ($\tau = 0.9$)	23.84	13.03	18.01	20.66	22.68	20.88	22.64	21.62	19.69	24.14	23.05	20.99	17.50	20.67
		Meta-set-Linear-1	1.23	10.05	3.32	4.54	2.31	3.69	1.41	2.85	3.51	0.22	1.58	3.7	3.31	3.21
		Meta-set-Linear-2	19.71	2.54	6.2	8.3	15.01	11.22	13.24	13.79	8.89	18.86	12.64	11.13	3.54	11.16
		Meta-set-Linear-3	29.07	6.7	15.46	17.59	24.36	20.54	22.55	23.13	18.18	28.23	21.95	20.45	12.77	20.08
		Meta-set-NN-1	0.54	25	13.3	10.94	3.75	7.87	3.92	2.62	10.86	0.95	5.1	5.04	16.32	8.17
		Meta-set-NN-2	18.26	0.11	5.29	7.93	16.83	13.69	15.28	15.53	8.25	19.58	14.13	14.81	2.68	11.72
		Meta-set-NN-3	29.23	14.15	15.89	18.11	27.32	25.37	25.23	24.23	18.8	29.07	24.46	24.08	13.5	22.26
		Acc_{bucket}	5.67	7.86	12.39	10.69	7.41	9.92	6.82	8.32	11.13	5.75	7.43	10.51	10.94	8.83
		$Acc_{confident}$	5.80	7.11	7.42	10.98	8.14	7.66	6.06	7.32	7.33	5.68	6.31	6.68	10.12	7.43
		Acc_{Aries}	0.07	0.37	2.49	0.15	0.36	1.13	0.38	0.50	1.90	0.03	0.56	1.92	0.41	0.79

is not enough, especially on the complex task (e.g., Tiny-ImageNet). The average difference of Acc_{bucket} and $Acc_{confident}$ of Tiny-ImageNet is greater than 7%, which is a very big bias. To understand why Acc_{Aries} works, we check the results of Acc_{bucket} and $Acc_{confident}$ separately and find that, generally, the Acc_{bucket} is over-estimation (48 out of 52 cases) while the $Acc_{confident}$ is under-estimation (48 out of 52 cases). We conjecture that this is because the learned decision boundary can not thoroughly work well on the data that have shifted distribution. The potential reason is that, in fact, the performance of the model on the high confident data (LVR is 1) of the shifted data can be lower than the original test data, e.g., 99.57% (original test data) vs 95.10% (Brightness data) of CIFAR-10-VGG16 model. The data with high confidence have a large proportion over all the data (e.g., there are 7969 data whose label consistent time is 50 for CIFAR-10-VGG16). Then, the results (line 4 in algorithm 3) can be overestimated. On the other hand, there are more data that the model has low confidence in but are still correctly predicted, e.g., 43, 51, 44 (Brightness data) vs 24, 27, 18 (original test data) when LVR times are 0.6, 0.62, and 0.64 of CIFAR-10-VGG16 model. This can make the size of high confident data in the shifted set as well as the $Acc_{confident}$ under-estimation. However, Acc_{Aries} finally averages the under- and over-estimation and produces a more precise accuracy. In the remaining experiments, we only report the results of Acc_{Aries}.



(a) CIFAR10, ResNet20 (b) CIFAR10, VGG16

(c) ImageNet, ResNet (d) ImageNet, DenseNet

Figure 7.4: Comparison with test selection methods. Transformation: contrast.

Comparison with baselines. First, we compare *Aries* with labeling-free methods. From Table 7.5 we can see that *Aries* outperforms the different configurations of the two baselines in most cases (50 out of 52). Besides, considering the average results, *Aries* can always stand out. Then, we compare *Aries* with test selection methods. Fig. 7.4 presents the results of shifted test sets with the contrast transformation. Considering the results produced by each test selection method, we found that there are some conflicting conclusions compared to the original papers. For example, [7] reports that PACE outperforms CES in its evaluation settings. However, from our results, only in CIFAR-10-VGG16-Contrast, PACE significantly outperforms CES. Under other settings, the results of these two methods fluctuate greatly. The same conflict occurs in random selection. In our evaluation, the existing well-designed test selection methods cannot consistently perform better than the random selection. This phenomenon reflects that, the evaluation of existing test selection methods for accuracy estimation is insufficient. Distribution shifts in data should be considered.

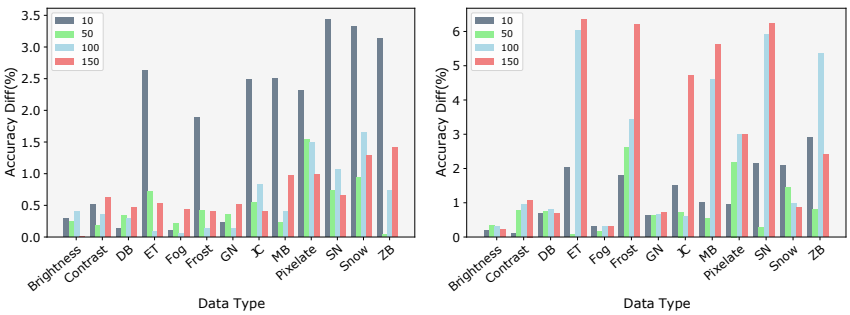
By comparison, *Aries* achieves competitive results with test selection methods. Although in some situations, selection-based methods achieve better results than *Aries*, e.g., in CIFAR-10-ResNet20-Brightness, when the labeling budget is 90, CES can estimate the accuracy more precisely. Overall, our technique performs the best in most cases (96 out of 128). Besides, *Aries* achieves more stable performance than the selection-based methods. For example, in CIFAR-10-ResNet20-Brightness, the estimation bias of CES can vary from 0.01% to 2.44% by using different labeling budgets, which could waste human resources while obtaining unexpected results.

Answer to RQ1: *Aries* estimates the model accuracy with a slight bias ranging from 0.03% to 2.60%. In addition, *Aries* outperforms labeling-free methods in 50 out of 52 cases and test selection-based methods in 96 out of 128 cases.

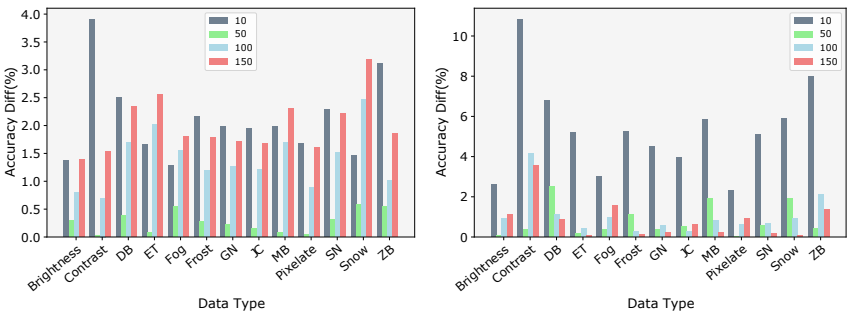
7.4.2 RQ2: Influencing Factor Study

Next, we explore how different configurations and settings affect the performance of *Aries*. As mentioned in Section 7.2.3, there are three main factors we need to consider, the number of buckets (n in algorithm 3), the dropout rate of the dropout layer, and the distance approximation method.

Number of Buckets. Fig. 7.5 presents the results of the accuracy estimation using *Aries* by different settings of the bucket number. The first conclusion we can draw is that, this factor has quite an impact on the results. For instance, in ImageNet-DenseNet121-Contrast, using 10 buckets can increase the accuracy difference by almost 10% than using 50 buckets. However, it's clear that there is no such setting that performs consistently better than others in all datasets and models. For example, in CIFAR10-VGG16, $n = 10$ is a relatively good setting. However, in CIFAR10-ResNet20, $n = 10$ is the worst among the four settings which means the setting of *Bucket* number is highly datasets and model-dependent. But if we check the more detailed values, we can see that, in total, in 5, 28, 6, and 13 cases, using 10, 50, 100, and 150 buckets can achieve the best estimation results, respectively. And on average, the differences between the estimated accuracy and the real accuracy are 2.62%, 0.61%, 1.39%, and 1.61%, respectively. Therefore, for the number of buckets, even though there can be no best setting, 50 is recommended among the studied settings. **Conclusion:** The number of *Bucket* highly impacts the performance of *Aries*. However, this hyperparameter setting is dataset and model-dependent. Thus, users should set this number according to the real use cases. $n = 50$ is a default setting of *Aries*.



(a) CIFAR10-ResNet20 (b) CIFAR10-VGG16



(c) ImageNet-ResNet101 (d) ImageNet-DenseNet121

Figure 7.5: Accuracy difference (%) between the real accuracy and the estimated accuracy by using different *Bucket* numbers.

Dropout Rate. Table 7.6 presents the difference between the real and estimated

Table 7.6: Difference (%) between the real and estimated accuracy by *Aries* using different dropout rates. DR: Dropout rate.

Dataset	DNN	DR	Brightness	Contrast	DB	ET	Fog	Frost	GN	JC	MB	Pixelate	SN	Snow	ZB	Avg.
CIFAR-10	ResNet20	**0.1**	0.31	1.27	0.10	4.79	0.44	3.33	0.08	3.91	5.82	3.20	5.60	4.84	6.40	3.08
		0.2	0.26	0.94	0.03	3.53	0.39	2.64	0.11	3.12	3.99	2.66	4.43	4.03	4.67	2.37
		0.3	0.15	0.45	0.37	2.45	0.20	1.88	0.33	2.39	2.58	2.29	3.40	3.21	3.04	1.75
		0.4	0.19	0.14	0.29	1.55	0.07	1.16	0.30	1.75	1.28	1.86	2.46	2.47	1.67	1.17
		0.5	0.25	0.18	0.34	0.72	0.21	0.42	0.35	0.55	0.23	1.54	0.74	0.94	0.04	0.50
		0.6	0.27	0.58	0.37	0.51	0.35	0.36	0.27	0.42	1.37	1.33	0.52	1.42	1.48	0.71
		0.7	0.38	1.22	0.55	2.38	0.90	1.50	0.48	0.34	3.14	0.92	0.61	0.80	3.67	1.30
		0.8	0.26	2.68	0.71	5.13	1.56	3.33	0.56	2.22	5.91	0.16	2.36	0.73	7.00	2.51
		0.9	0.28	5.70	1.63	10.76	3.91	8.06	1.57	6.40	11.21	1.65	6.66	5.30	14.16	5.95
	VGG16	**0.1**	0.09	0.03	0.24	0.15	0.11	0.56	0.09	0.99	0.23	0.55	0.82	0.79	0.55	0.40
		0.2	0.25	0.23	0.13	1.37	0.03	0.89	0.25	0.56	0.70	0.31	0.73	1.03	1.85	0.64
		0.3	0.08	0.09	0.50	2.29	0.02	1.78	0.61	1.47	1.09	0.83	2.09	2.03	2.97	1.22
		0.4	0.04	0.29	0.58	3.28	0.13	3.12	0.46	2.41	2.02	1.41	0.52	0.00	4.01	1.41
		0.5	0.33	0.77	0.74	0.08	0.16	2.60	0.63	0.73	0.53	2.19	0.27	1.45	0.81	0.87
		0.6	0.07	0.37	1.44	6.66	0.13	6.15	0.83	4.96	5.28	3.46	6.20	8.14	8.01	3.98
		0.7	1.37	1.19	0.13	11.16	0.93	9.28	1.00	7.87	9.80	4.50	7.65	11.22	12.75	6.07
		0.8	2.14	7.92	3.09	17.70	6.21	14.39	1.31	14.03	16.51	6.66	10.76	15.33	22.79	10.68
		0.9	4.44	18.12	20.54	27.27	1.19	16.61	7.01	20.87	25.63	0.79	10.75	24.61	30.86	16.05
Tiny-ImageNet	ResNet101	**0.1**	3.12	7.44	5.14	4.86	3.74	4.46	5.02	4.09	4.90	3.85	4.97	4.02	6.18	4.75
		0.2	2.36	5.24	3.33	3.01	2.25	3.01	3.60	2.72	2.78	2.76	3.42	2.31	4.19	3.15
		0.3	1.37	3.23	1.73	1.36	0.75	1.81	2.02	1.23	1.42	1.57	1.78	0.82	2.57	1.67
		0.4	0.18	1.29	0.11	0.60	0.22	0.37	0.29	0.24	0.06	0.36	0.06	0.53	0.56	0.37
		0.5	0.29	0.03	0.38	0.08	0.55	0.28	0.23	0.15	0.08	0.05	0.31	0.59	0.54	0.27
		0.6	1.11	1.44	1.69	2.70	2.01	1.65	1.44	1.75	2.14	1.30	2.36	3.08	1.68	1.87
		0.7	1.82	3.10	2.92	3.36	2.82	2.66	2.94	3.15	2.72	2.56	3.80	4.26	3.37	3.04
		0.8	0.16	3.95	3.30	4.03	2.32	4.16	3.03	2.52	2.83	1.86	4.04	3.99	3.36	3.04
		0.9	5.81	2.88	3.60	8.32	4.12	1.02	1.09	1.13	1.86	3.06	3.37	6.34	0.83	3.34
	DenseNet121	**0.1**	0.24	5.41	2.15	1.40	0.28	1.03	1.04	0.30	1.45	0.24	1.26	1.76	3.20	1.52
		0.2	0.09	0.34	2.56	0.05	0.55	1.11	1.66	0.38	1.49	0.17	1.40	1.78	1.36	1.00
		0.3	0.09	1.52	2.41	0.31	0.52	0.84	1.36	0.34	1.62	0.14	1.62	2.08	1.09	1.07
		0.4	0.36	0.83	2.71	0.46	0.02	1.45	1.60	0.75	1.84	0.15	1.76	2.42	0.66	1.15
		0.5	0.07	0.37	2.49	0.15	0.36	1.13	0.38	0.50	1.90	0.03	0.56	1.92	0.41	0.79
		0.6	0.37	0.78	2.34	0.42	0.38	1.30	1.77	0.63	1.85	0.06	1.85	1.65	0.33	1.06
		0.7	0.21	0.55	2.39	0.01	0.60	1.00	1.08	0.35	1.57	0.35	2.01	1.92	1.07	1.01
		0.8	0.46	0.39	2.15	0.15	0.37	0.98	1.35	0.33	1.43	0.62	1.33	1.64	1.45	0.97
		0.9	0.08	0.65	2.51	0.29	0.60	0.97	1.16	0.23	1.58	0.01	1.95	1.94	1.52	1.04

accuracy by *Aries* using different dropout rates. Similar to the study of the *Bucket* number, there is no dropout rate setting that can consistently outperform others. But still, a relatively better setting exists through a deeper analysis. In total, in 6, 5, 3, 10, 19, 4, 2, 6, and 1 cases, the dropout rate 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9 achieve the best estimation results, respectively. This indicates that the dropout rate = 0.5 is the best among all the studied settings in terms of achieving the most precise estimation. Then, looking into the average results, we draw a similar conclusion that when we set the dropout rate as 0.5, in 3 (out of 4) cases, the estimated accuracy is closer to real accuracy compared to other settings. Fig. 7.6 depicts the trend of the average difference between real and estimated accuracy by using different dropout rates (Column *Average* in Table 7.6). We can see that, the difference drops first when the dropout rate increases, and after the dropout rate reaches around 0.5, the difference increases. **Conclusion:** There is no dropout rate setting that always performs the best. We recommend using the dropout rate of around 0.5 for *Aries* to gain better results.

Mutant for Distance Estimation. Finally, since at each prediction time, the dropout model can be seen as a mutant of the original model, we explore if we can utilize model mutation for replacing dropout prediction to approximate the distance between data and decision boundaries.

Table 7.7 presents the results of the comparison between the two ways of approximating the distance between the data and the decision boundaries. When replacing the dropout with mutants (change M in Definition 1 to mutants), *Aries* still works in some cases. For CIFAR-10, *Aries* with mutants can still produce some acceptable results, e.g., in 10 cases, the difference is smaller than 1%. Compared to *Aries* with dropout, in 10 out of 26 cases, the mutant achieves better results. On average,

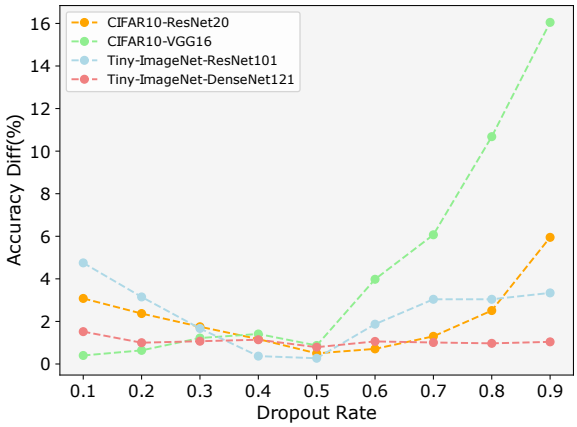


Figure 7.6: The trend of the average difference between real accuracy and estimated accuracy by using different dropout rates. The rate at 0.5 is a common turning point of the accuracy difference for all datasets and models.

Table 7.7: Comparison between dropout and mutant. Each value is the absolute difference between real and estimated accuracy (%).

Data Type	CIFAR-10				Tiny-ImageNet			
	ResNet20		VGG16		ResNet101		DesNet121	
	Mutant	Dropout	Mutant	Dropout	Mutant	Dropout	Mutant	Dropout
Brightness	0.12	0.25	0.01	0.33	4.49	0.29	6.71	0.07
Contrast	0.95	0.18	0.00	0.77	10.67	0.03	19.62	0.37
DB	0.24	0.34	0.15	0.74	7.80	0.38	14.08	2.49
ET	3.30	0.72	1.19	0.08	7.72	0.08	11.74	0.15
Fog	0.51	0.21	0.14	0.16	6.02	0.55	8.43	0.36
Frost	2.97	0.42	2.36	2.60	7.08	0.28	11.25	1.13
GN	0.25	0.35	0.21	0.63	7.28	0.23	9.94	0.38
JC	3.38	0.55	2.68	0.73	6.39	0.15	9.52	0.50
MB	4.04	0.23	1.17	0.53	7.34	0.08	12.77	1.90
Pixelate	2.80	1.54	1.56	2.19	5.43	0.05	6.43	0.03
SN	4.49	0.74	3.00	0.27	7.34	0.31	10.67	0.56
Snow	3.66	0.94	2.95	1.45	6.67	0.59	11.61	1.92
ZB	4.50	0.04	1.26	0.81	9.28	0.54	15.50	0.41
Avg.	2.40	0.50	1.28	0.87	7.19	0.27	11.41	0.79

there are only 1.9% and 0.42% effectiveness gaps between these two ways. However, the performance of *Aries* with mutants becomes worse in Tiny-ImageNet, and the difference increases significantly compared to using dropout. In all the cases, *Aries* with mutants performs worse than with dropout. This phenomenon indicates that *Aries* with mutants can only work on simple datasets and models. This is reasonable because, at each prediction time, the status of the dropout model might appear in the training process due to its design nature [162]. Thus, it can reflect the learned decision boundary more precisely. However, the post-training model mutation randomly modifies the model, which could totally change the learned decision boundary even if it maintains the accuracy. **Conclusion:** *Aries* with using mutants can achieve similar accuracy estimation results with using dropout in CIFAR-10 dataset, while fails in Tiny-ImageNet dataset. It needs more careful mutation operator selection and hyperparameter tuning to ensure the decision boundary does not change too much after model mutation.

Answer to RQ2: The number of buckets and dropout rate affect the performance of *Aries*. 50 and 0.5 are the recommended settings, respectively. Simply replacing the dropout with mutants can work on the simple dataset (CIFAR-10) but fails on the complex dataset (Tiny-ImageNet).

7.5 Discussion and Threat to Validity

7.5.1 Limitations & Future Directions

Limitations. 1) The first potential limitation is that *Aries* might suffer from adversarial attacks [163]. A strong adversarial attack method can control the distance between the adversarial examples and the decision boundaries by pushing the original data close to or far away from the decision boundary, which invalidates our learned boundary information. However, adversarial attack is a common concern for all methods. For example, for the output-based methods (PACE and CES), white-box adversarial attacks can be designed to change the output of the neurons to force these methods to select useless data to label. How to defend against adversarial attacks is an open problem. 2) The second limitation comes from our assumption that we already have some labeled test data. In general, this assumption can stand. However, in extreme cases where only the model and new unlabeled data are available, we still need to undertake data labeling.

Future directions. 1) We only utilize the original labeled test data to gain the decision boundary information, and it works well in our evaluation subjects. There could be a way to do data augmentation based on the labeled data and increase the data space we have. In this way, we can learn more precise boundary information and, therefore, make better accuracy estimations for the new unlabeled data. 2) Although dropout uncertainty is the widely studied uncertainty method and works well in our technique, the mutant prediction is the closest way to the dropout prediction. Some other uncertainty methods can be used to approximate the distance between data to the decision boundaries, e.g., DeepGini [23]. We plan to study more methods and explore if there is a better way to replace dropout prediction. On the other hand, adversarial attacks can be used to achieve the same goal [164]. 3) Maybe more interestingly, we tend to explore if *Aries* can be used in other types of datasets and models, e.g., models for code learning.

7.5.2 Threats to Validity

The internal threats to validity are the implementations of our technique, the baselines, and mutation operators as well as the preparation of new unlabeled data. Our technique is simple and easy to implement and its core part is using pure Python. Also, we release our code for future study. All the implementations of the baselines and the mutation operators are from the original papers. For the new unlabeled data, to reduce the influence of parameter settings (e.g., which levels of brightness should be added), we directly reuse the released datasets that are widely studied in the literature.

The external threats to validity come from the selected datasets and models for our evaluation. For the dataset, we use two commonly studied ones from the recent research [165, 166, 167]. For each dataset, we build two different model architectures from simple to complex. Compared to the existing test selection works [7, 6] which stop by ResNet-50, our considered model architectures are more complex (ResNet101

and DenseNet121). Besides, another threat that comes from the model could be model calibration (roughly speaking, the diversity of models) [168]. Actually, our evaluation involves both well-calibrated and poorly-calibrated models. Indeed, the Predicted score-based method can witness model calibration to some extent. For CIFAR-10-VGG16, this method (with $\tau = 0.9$) reveals that around 90% of data have $> 90\%$ confidence, thus, the model is over-confidence. For a similar reason, Tiny-ImageNet-ResNet101 is under-confident. For the new unlabeled data, we also follow the previous works [169, 170, 171] that evaluate the model robustness to prepare our test sets. Even though collecting unlabeled data in the wild is a good way to further evaluate our method. It is not easy to collect and label massive new data. In this paper, we believe the 13 transformation techniques we used to simulate distribution shifts can achieve high data diversity.

7.6 Conclusion

We proposed *Aries* to automatically estimate the accuracy of DNN models on unlabeled data without labeling effort. The main intuition of *Aries* is that a model should have similar prediction accuracy on the data that have similar distances to the decision boundaries. Specifically, *Aries* employs the dropout uncertainty to approximate the distance between data and decision boundaries and learns this boundary information from the original labeled test data to estimate the accuracy of new unlabeled data. Our evaluation of two commonly studied datasets, four DNN architectures, and 13 types of unlabeled data demonstrated that *Aries* can precisely predict the model accuracy. Besides, we demonstrated that *Aries* outperforms SOTA labeling-free estimation methods and test selection-based methods.

8 An Empirical Study on Data Distribution-Aware Test Selection for Deep Learning Enhancemen

In this chapter, we first conduct a systemically empirical study to reveal the impact of the retraining process and data distribution on model enhancement. Leveraging the insights gleaned from this investigation, we propose a novel distribution-aware test (DAT) selection metric for model retraining.

Contents

8.1 Introduction

Deep Neural Networks (DNNs) are increasingly integrated into large software systems in various applications, such as face recognition [172], autonomous vehicles [173], speech recognition [174], video gaming [149], and so on. Despite the impressive success and great potential of DNNs, there are crucial accidents caused by quality issues of deep learning (DL) systems, e.g., Tesla/Uber accidents [175]. Therefore, similar to traditional software products, DNNs are required to undertake careful testing to check whether they match the expected requirements for reliable deployment. In practice, DNNs are mostly tested on a set of examples – the *test set* – that is extracted from the same dataset as the training set. As a result, by default, the test set and training set follow the same data distribution.

However, in real-world applications, Deep Learning (DL) systems face an important hurdle: the effectiveness (e.g. prediction accuracy) of the embedded DNN declines over time due to changes in data distribution. These distribution shifts [176] originate from multiple causes, e.g., changes in user behavior, seasonal data patterns, benign alterations in the inputs, etc. In such cases, software engineers have no choice but to manually re-engineer the DNN (i.e. design the architecture, set the hyperparameters, and train on the data anew). These re-engineering activities require considerable human and computational effort that is akin to the original production of the model. Distribution shift, therefore, constitutes one of the most important obstacles to the widespread dissemination of DL.

Similar to the general problem of software maintenance in conventional software engineering, distribution shift concerns enhancing the capability of the ML model to deal with unseen inputs. With the rapid growth of data that could follow a different data distribution, DL models may exhibit a misleading sense of achieving high performance on the original test data while having unexpected performance on the new data. Therefore, DL systems – in particular, the DNNs that are the essential backbone of these systems – also need to be evolved upon having the massive amount of collected new test data for continuous enhancement.

Fortunately, DL systems do not need to be re-engineered each time a distribution shift occurs but can rather cope with such shifts through a development strategy that promotes incrementality. Common strategies to combat drifts include retraining the DL model, that is, updating the DNN weights through additional training epochs using the new data. The retraining process can be entirely automated and, therefore, can avoid the heavy human and computational overhead of complete re-engineering. (Re)Training a DNN requires labels of the collected data to calculate the loss information and guide the tuning of the model weights. However, data labeling is another important practical overhead. The reason is that although collecting massive new data (usually raw and unlabeled) is cheap and easy, labeling all of them is often manual, expensive, and prohibitively time-consuming. For example, labeling the first version of the ImageNet dataset took groups of people more than 3 years [86]. Particularly, the manual task of labeling can be more challenging in specific applications, when domain-specific knowledge is required.

Test selection refers to the area of research concerned with selecting, from a large set of unlabelled data, those data that are more likely to reveal errors in a given DNN [24]. Research has recently developed selection metrics to address this problem [177, 178, 95, 179, 180] as well as reduce the labeling effort. Once fault-revealing data have

been found and labeled, the same data can be used to retrain the model (removing the errors that these data represent) and, thereby, improve its generalization. While these metrics have demonstrated their potential to test and improve DNNs, we observed fundamental and experimental gaps that we aim to address in this paper:

1. Utilization of two different retraining processes. The retraining process plays a key role in the model enhancement, which leads the model to learn new information while keeping the original knowledge. However, in existing studies, two different retraining processes are used for model enhancement and the impact of each process is still unclear and not explored. Taking three state-of-the-art metrics as an example, the Multiple-Boundary Clustering and Prioritization (MCP) [178] and the surprise adequacy guided metric [95] retrain a DNN using only this subset. On the contrary, DeepGini [177] uses both the original training data and this subset.

2. Unaware of data distribution shift. The shift of data distribution refers to the phenomenon that the distributions of training and test data are different, such as the images taken under different brightness. Usually, the data following the same or a different distribution are regarded as in-distribution (ID) or out-of-distribution (OOD) data, respectively. The distribution shift can be divided into two types, 1) synthetic distribution shift which comes from the computer-generated perturbation; 2) natural distribution shift which comes from unseen and unperturbed data. Data distribution has been proven to be critical in deep learning testing, especially for practical deployment of DL models [47, 48]. However, this factor is not considered in existing test selection metrics.

3. Evaluated by narrow experimental setups. We observe that the effectiveness of existing selection metrics for model retraining is insufficiently evaluated. For instance, MCP is only evaluated on a combination of original test data (80%) and new data (20%), while DeepGini only selects data from the new data (100%) and retrains the model accordingly. The other combinations of data are uncovered and should be considered in the evaluation.

To elaborate on and address these limitations, in this paper, we conduct an empirical study to evaluate existing selection metrics under various data distribution shifts and answer the following three research questions:

RQ1: Which retraining process achieves better model enhancement?

RQ2: How effective are different test selection metrics under different data distributions for model enhancement?

RQ3: Concerning data distribution and class bias, what are the characteristics of the data selected by different metrics?

Overall, our empirical study evaluates 6 selection metrics over 5 datasets (including 3 image datasets and 2 text datasets) and 2 DNN architectures for each dataset (including both feed-forward neural networks (FNNs) and recurrent neural networks (RNNs)). In total, we retrained 71280 models. By investigating the above research questions, we found that retraining using both the original training data and selected data achieves better results for model enhancement. Moreover, we observed that using this retraining process, existing selection metrics perform differently under

different data distributions. For example, when OOD data are more than 70% in the new set, random selection performs surprisingly the best. Besides, we found that class bias is another potential characteristic in addition to data distribution for data selection. Based on these findings, we further propose a distribution-aware test (DAT) selection metric to alleviate the impact of distribution shifts on model retraining. The key idea of DAT is to **select uncertain and representative data from the ID and OOD sets, respectively**. In detail, we first utilize an OOD detector to split the new data into the ID set and OOD set. Afterward, for the ID set, DAT selects the most uncertain data which follow the same distribution as the training data but are not well learned by the model. For the OOD set, DAT selects the most representative data, which means the selected data can represent the whole set. To demonstrate the effectiveness of our metric, we conduct experiments to answer the following two research questions. The experimental results show that DAT achieves the best performance among all the existing metrics.

RQ4: Under synthetic distribution shifts, how effective is DAT for model enhancement?

RQ5: Under natural distribution shifts, how effective is DAT for model enhancement?

In summary, the main contributions of this paper are:

- To the best of our knowledge, we are the first to conduct a systemically empirical study of investigating how the retraining process and data distribution impact the test selection for model enhancement.
- This is the first study that analyzes and explores the characteristics of data selected by different metrics in terms of both data distribution and class bias.
- We propose the first distribution-aware test selection metric (DAT), which can reduce the impact of data distribution on model enhancement. Besides, we release our implementation and datasets for future use and research.[1].

8.2 Objectives and Problem Formulation

Let us consider a N-class classification task over data $X \subseteq \mathbb{R}^d$ and labels $Y \subseteq \mathbb{Z}$. Let $f : \mathbf{x} \to y$ refer to a DNN trained on $X^{in} \subset X$, with $\mathbf{x} \in \mathbf{X}$ and $y \in Y$. We denote the distribution of the data X^{in} as $\mathcal{D}_{in}$ and we refer to these data as the *ID data*. Now let X^{out} be a set of data that follow a mixture of distributions $\mathcal{D}_{in}, \mathcal{D}_{out}$ where $\mathcal{D}_{out}$ is an arbitrarily complex (possibly a mixture) distribution that differs from $\mathcal{D}_{in}$. That is, X^{out} is a data sample that results from a distribution shift from $\mathcal{D}_{in}$ to $\mathcal{D}_{in}, \mathcal{D}_{out}$. We furthermore assume that X^{out} is unlabelled and we name these data the *OOD data*.

Our goal is to decrease the computational and human effort to improve models when distribution shift occurs. We aim to maximize the performance (e.g. accuracy) of the DNN on some $X^{out}_{test} \subset X^{out}$. We assume that we are allowed to change neither the architecture nor the hyperparameters of the DNN. Instead, we follow the straightforward and low computation-cost method that consists to retrain the model for an additional number n of epochs with an independent sample $X^{out}_{train} \subset X^{out}$ that has no overlap with X^{out}_{test}. Given that we aim to minimize labeling cost, we also want $|X^{out}_{train}|$ to be under a predefined data budget b.

To address this challenge, we empirically investigate two key factors that may affect the effectiveness of retraining: the retraining process and the selection metric

[1] https://github.com/code4papers/DAT

– i.e. the metric used to select X^{out}_{train} from X^{out}. Our analysis of the literature has revealed two types of retraining processes: retrain with X^{t}_{train} only or with a mixture of X^{out}_{train} and X_{in}. As for selecting X^{out}_{train}, we consider selection metrics that have been proposed in the literature and have also been used for retraining [177, 178, 95, 179, 180].

8.3 Empirical Study Methodology

First of all, to answer the first three research questions, we conduct a comprehensive empirical study to explore how retraining processes and data distribution affect the effectiveness of selection metrics for model retraining. This study provides the motivation for our proposed distribution-aware selection metric (Section 8.5).

8.3.1 Study Design

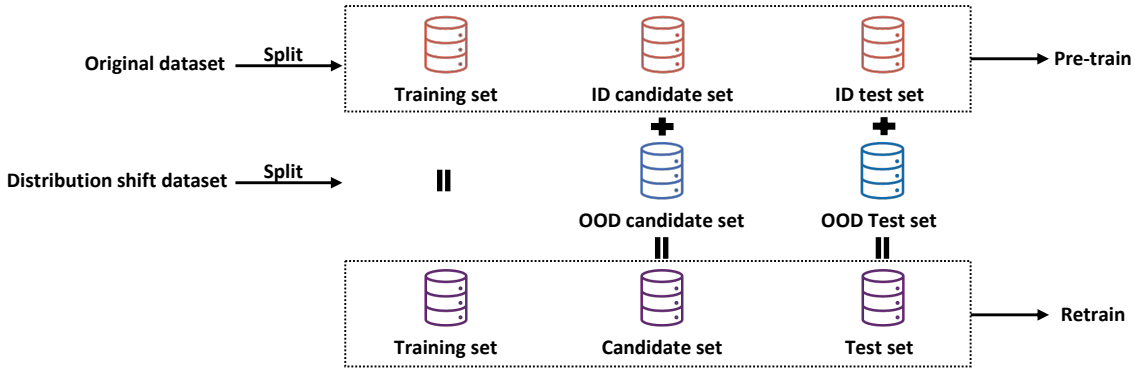


Figure 8.1: Procedure of data preparation. ID and OOD are short for in-distribution and out-out-distribution, respectively. All candidate sets are unlabeled, and the others are labeled.

To conduct the empirical study, we first prepare the data as shown in Figure 8.1. Given a dataset, we randomly split it into three separate sets, training set, ID candidate set, and ID test set, to build pre-trained DNNs. Afterward, we partition the distribution shift (OOD) dataset into the OOD candidate and test sets. Please refer to Section 8.3.4 for details of obtaining distribution shift datasets. Finally, we combine ID and OOD data with a certain ratio to simulate different distribution shifts. For instance, 10% ID + 90% OOD indicates that the new data has a dramatic shift where 90% data are unseen by pre-trained DNNs. In our study, we use 11 different combinations with the ratio ranging from 0% to 100% at a 10% interval. The candidate set represents new unlabeled data for selection and retraining, and the test set follows the same distribution as the candidate set for performance evaluation.

Figure 8.2 gives an overview of our empirical study. 1) We first prepare pre-trained models for each dataset, 2) then utilize different selection metrics to select and label data. Next, 3) we use the selected data to retrain the pre-trained model with another few epochs. Finally, 4) we test the retrained models on both the ID and new test sets.

One factor that could highly affect the performance of the retrained model is the retraining process. In the literature, there are mainly two processes for model retraining. One is to retrain using only the selected data [95, 178]. The other is using both the training and selected data [177]. To answer **RQ1**, we apply both processes

separately to produce two retrained DNNs. Next, we test the retrained models on test sets and compute their performance. Later, based on the findings of **RQ1**, we will apply the better retraining process to analyze how the data distribution would affect the effectiveness of each selection metric for model retraining and answer **RQ2**. In this phase, we only consider the test accuracy of retrained models. Furthermore, we investigate the properties of selected data by different selection metrics to answer **RQ3**.

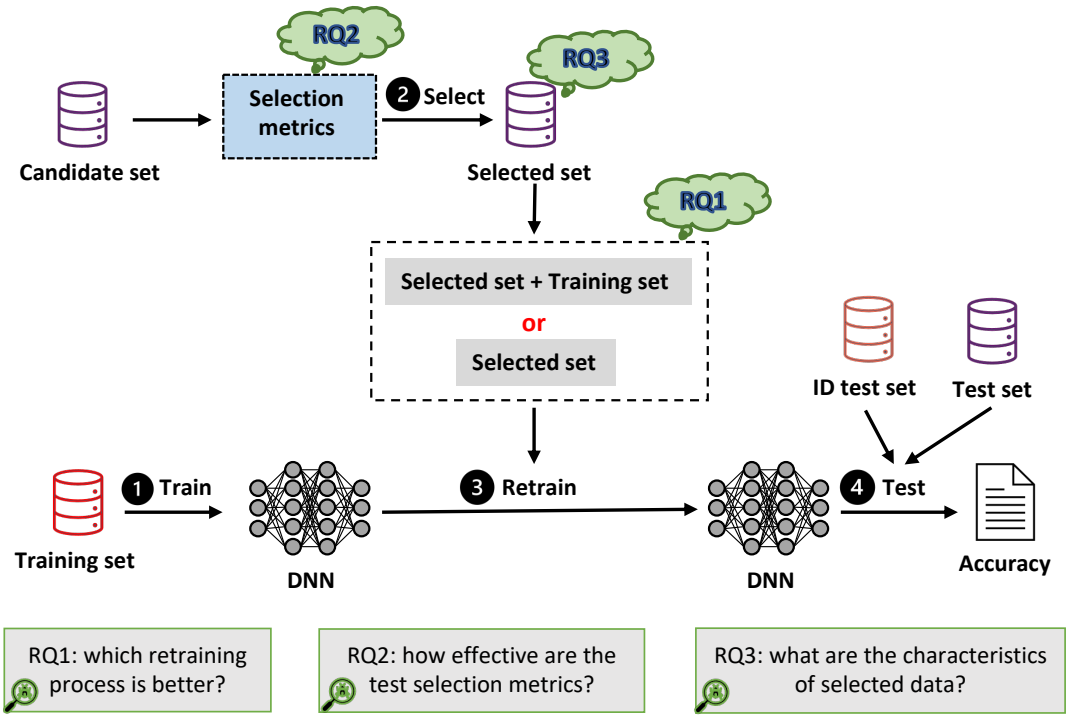


Figure 8.2: Overview of our empirical study.

8.3.2 Datasets and DNNs

In our empirical study, we consider 5 publicly available datasets, MNIST [73], Fashion-MNIST [181], CIFAR-10 [75], IMDb [103], and Newsgroups [182]. MNIST is a collection of grayscale images of hand-written digits, e.g., 1, 2. Fashion-MNIST includes grayscale images of fashion products, e.g., coat, shirt. CIFAR-10 contains color images, e.g., airplane, bird. IMDb is a dataset containing movie reviews that are widely used for sentiment analysis (i.e., positive or negative). Newsgroups is a text dataset that includes 20 different newsgroup subjects, e.g., space, baseball. For MNIST, Fashion-MNIST, and CIFAR-10, we randomly pick 10000 data from the training set as the candidate set. For IMDb and Newsgroups, we randomly collect 5000 and 4000 data from the training set as the candidate set, respectively. For each dataset, we use two different well-known DNN models in previous research. For the image datasets, we consider the famous convolutional neural networks (CNNs), for example, LeNet and ResNet. Since recurrent neural networks (RNNs) are good at handling sequential data, we utilize embedding layers to encode the text into vectors first, then we use RNNs to process the vectors and predict sentiment results. Besides, we follow [183] to build the fully connected neural network for the Newsgroups dataset. Hence, our study covers image and text data, feed-forward, and recurrent

neural networks. All the detailed model architectures and hyper-parameters are available on our project website 1. Table 8.1 shows details of the datasets and DNNs. We measure model performance in terms of accuracy, as it is the metric originally used for the tasks and datasets that we study. Note that since we do not use all the training data to train the model, the test accuracy of each model may not achieve the state-of-the-art.

Table 8.1: Datasets and DNN models. "**Test accuracy**" is the accuracy (%) of the ID test set (see Figure 8.1).

Dataset	Data Type	#Training	#Test	#Classes	DNN	#Layers	#Parameters	Test accuracy (%)
MNIST	Image	60000	10000	10	LeNet-1	5	3246	97.91
					LeNet-5	7	107786	98.90
Fashion-MNIST	Image	60000	10000	10	LeNet-1	5	3246	87.29
					LeNet-5	7	107786	90.29
CIFAR-10	Image	50000	10000	10	ResNet-20	20	274442	85.79
					NiN	23	972658	87.16
IMDb	Text	25000	5000	2	LSTM	5	2694206	85.61
					GRU	5	2661694	86.46
Newsgroups	Text	4000	1000	20	NN1	2	450650	86.70
					NN2	3	452600	81.30

8.3.3 Selection Metrics

Various selection metrics have been proposed and evaluated for data prioritization and data labeling effort reduction. In this study, we choose 4 metrics (MCP, DeepGini, CES, and DSA) proposed in the SE community. Note that MCP, CES, and DSA have been evaluated as the best metrics in a recent study [178] compared with the others, such as the likelihood-based surprise adequacy (LSA) [95] and adaptive active learning (AAL) [184]. DeepGini is a newly proposed method for enhancing the performance of DNNs. Additionally, we take the random selection metric as the baseline. Given that the task of active learning within each stage is similar to test selection, the most basic and popular metric, Entropy, [89] is also considered for comparison. We briefly introduce each metric as follows.

Throughout the paper, we use $p_i(\mathbf{x})$, $0 \leq i \leq N$ to represent the predicted probability of $\mathbf{x}$ belonging to the ith class.

1) Random Random selection is a basic and the simplest selection method. It draws data directly from the given set regardless of the model's behavior. Each data is randomly selected, namely, has the same probability of being chosen.

2) Multiple-Boundary Clustering and Prioritization (MCP) MCP [178] selects test data limited in decision boundary areas. Concretely, it proceeds in three steps. First, the DNN model runs on each test sample to give a sequence of output probabilities. Second, MCP conducts a boundary area clustering to divide the data into different clusters. A cluster (the boundary area between two classes) is formed according to the top two classes of test data. Besides, for each test data, MCP computes its priority in its belonging cluster as the ratio of the probability of the first class to the probability of the second class. Finally, test data with high priorities are evenly selected from each non-empty cluster. The intuition behind MCP is that if the top-2 probabilities of a test sample are close, this sample is close to the decision boundary between the corresponding two classes.

3) Cross Entropy-based Sampling (CES) The main idea of CES [179] is to select a subset of test data that can maximally represent the distribution of the entire test dataset via the cross entropy. More specifically, this subset should have the minimum

cross entropy with the entire test dataset. To solve this optimization problem, CES utilizes a similar algorithm to the random walk [185]. It starts with a random subset T (smaller than the budget) with a few test data, then repeatedly enlarges T by merging another subset P that is randomly selected and has the minimum cross entropy with T.

4) Distance-based Surprise Adequacy (DSA) DSA [95] is an adequacy criterion that aims at measuring how surprising a test sample is to a DNN model concerning the training data. It computes the surprise adequacy by the Euclidean distance between the model's behaviors represented by the activation traces of the test sample and the training set. Finally, the data with high adequacy are selected.

5) DeepGini Similar to Entropy, DeepGini [177] also selects the most uncertain data using the output probabilities by:

$$\arg\max_{\mathbf{x}\in X}\left(1-\sum_{i=1}^{N}\left(p_i\left(\mathbf{x}\right)\right)^2\right) \tag{8.1}$$

6) Entropy-based metric (Entropy) As a widely used information-theoretic metric, entropy, also known as Shannon entropy [186], measures the average level of information required to obtain a possible prediction. In other words, it calculates the uncertainty for a DNN model to output a prediction. Based on this concept, Entropy [89] selects the test data that have the maximum uncertainties, and its formal definition is:

$$\arg\max_{\mathbf{x}\in X}\left(-\sum_{i=1}^{N}p_i\left(\mathbf{x}\right)\log p_i\left(\mathbf{x}\right)\right) \tag{8.2}$$

Most of the aforementioned metrics (MCP, CES, DeepGini, and Entropy) are only designed for classification tasks since they require the output probability of each class in their methodologies. The only exception is DSA, which also works for regression tasks. Our metric DAT is also designed for classification tasks – one objective of DAT is to collect data with balanced classes. Therefore, in our study, we only focus on the classification tasks. Nonetheless, to the best of our knowledge, our study is the largest one that considers both image and text classification tasks with both synthetic and natural distribution shifts.

8.3.4 OOD Data Preparation

In our study, we consider two types of distribution shift, synthetic and natural. Both are widely studied in recent works [187, 183].

8.3.4.1 OOD data with synthetic distribution shift

Synthetic distribution shift comes from the computer-generated perturbation. In the literature [178, 47, 18, 188], there are two types of image mutation methods to generate noise data: image transformation [189] and adversarial attack [190]. Table 8.2 describes the six image transformations and the two adversarial attacks used in our study. Image transformation applies basic geometric transformations to mimic different real-world conditions such as changing the contrast or brightness of images and rotating the camera. Here, we consider transformations that are common in the real world and whose relevance has been shown in previous studies [189, 178, 47]: rotation, shear, translation, scaling, brightness, and contrast. We follow [178, 47] to set up the parameters of these transformations, for example, for the MNIST scale, we

Table 8.2: Description of mutation operators

Type	Mutation Operator	Description
Transformation	Rotation	Rotate an image by a certain angle
	Shear	Shear an image horizontally
	Translation	Translate several pixels downright
	Scale	Change the size of an image
	Brightness	Adjust the brightness of an image
	Contrast	Adjust the contrast of an image
Attack	FGSM	Fast gradient sign method
	PGD	Project gradient descent

set the scale coefficient as 0.8. All the parameters of image transformations can be found on our project site 1. Adversarial attacks add an imperceptible perturbation into an image to mislead DNNs. These attacks have been associated with distribution shifts and can be useful to improve the generalization ability of ML models [191]. We use two of the most common attack algorithms, FGSM [191] and PGD [192]. We utilize the L_{inf} distance to calculate the perturbation with a commonly used [193, 192] maximum size of 0.3 (8/255) for MNIST and Fashion-MNIST (CIFAR-10).

To make sure that each mutation method (i.e. each image transformation and adversarial attack) introduces distribution shifts, we empirically show that there is a greater distribution difference (1) between the original training set and the original test set and (2) between the original training set and the mutated test set. If (2) is greater than (1), then it would mean that the mutations induce a distribution shift compared to the natural difference that is due to data generalization. To measure such distribution differences, we combine a state-of-the-art Outlier Exposure (OE) detector [194] (more details in Section 8.5.1) and Jensen-Shannon Divergence (JSD) score [195]. OE enables the identification of data that do not belong to a given distribution (in our case, the original training set determines the distribution). It assigns a score to each example, where a higher score means that the example is farther from the given distribution. To build the OOD detector, we need a baseline of out-of-distribution data (OOD) that are clearly not from the original distribution. In our case, to build the OOD detectors for MNIST, Fashion-MNIST, and CIFAR-10, we use, respectively, Fashion-MNIST, MNIST, and SVHN. The reason behind this choice [47] is that MNIST and Fashion-MNIST are black-and-white images, whereas CIFAR-10 and SVHN are colored. Once we have an OOD detector, we predict the score of the examples in the two test sets and build the corresponding two histograms. We calculate the JSD between the two histograms. JSD is an established metric for the dissimilarity between two probability distributions. A higher JSD indicates higher dissimilarity.

Table 8.3 lists the results. The JSD between the original training and test sets (at most 0.07) is much smaller than the JSD between the training and mutated sets (at least 0.21). For Fashion-MNIST, Some mutated sets have an even greater JSD score than the OOD sets, revealing that the shifts that mutation induces can be more significant than a shift to a completely different dataset. In conclusion, these results confirm that the used mutations are indeed appropriate to emulate distribution shifts.

8.3.4.1.1 OOD data with natural distribution shift Natural distribution shift comes from unseen environments. For the text datasets, it is easy to collect this

Table 8.3: JSD between training set and other sets

	Test	Brightness	Contrast	Rotration	Scale	Shear	Translation	FGSM	PGD	OOD
MNIST	0.05	0.77	0.52	0.61	0.62	0.57	0.56	0.73	0.65	0.77
Fashion-MNIST	0.05	0.62	0.36	0.48	0.53	0.55	0.52	0.56	0.38	0.44
CIFAR-10	0.07	0.21	0.50	0.51	0.57	0.47	0.47	0.40	0.25	0.60

kind of OOD data that targets the same task as the ID data, e.g., we can collect the movie reviews from different websites and groups of people. We obtain such datasets (IMDb and Newsgroups) from the baseline work [183] directly. Following the same settings as [183], for the IMDb dataset, we use the combination of customer reviews and movie reviews as the OOD data. For the Newsgroups, we randomly choose 10 groups as the ID data and 10 groups as the OOD data.

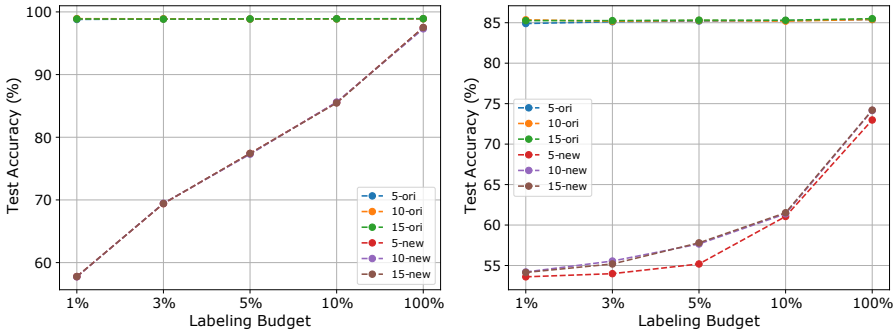
Table 8.4: Average test accuracy (%) of the test set (see Figure 8.1).

Distribution	**MNIST**		**Fashion-MNIST**		**CIFAR10**		**IMDb**		**Newsgroups**		**Average**
ID + OOD	**LeNet-1**	**LeNet-5**	**LeNet-1**	**LeNet-5**	**NiN**	**ResNet20**	**LSTM**	**GRU**	**NN**	**NN2**	
0% + 100%	28.65	36.23	22.62	20.19	51.05	46.90	68.36	67.58	0.40	6.90	34.89
10% + 90%	35.59	42.44	29.11	27.37	54.68	50.85	70.04	69.44	9.00	14.00	40.25
20% + 80%	42.40	48.66	35.51	34.33	58.35	54.69	71.78	71.56	17.50	21.90	45.67
30% + 70%	49.28	54.82	41.86	41.30	61.95	58.50	73.64	73.94	26.20	29.60	51.11
40% + 60%	56.19	61.03	48.31	48.02	65.48	62.51	75.48	75.56	35.20	37.50	56.53
50% + 50%	63.14	67.46	54.87	54.95	68.99	66.29	77.34	77.60	44.40	45.00	62.00
60% + 40%	70.14	73.67	61.36	61.95	72.74	70.27	79.04	79.46	53.10	52.30	67.40
70% + 30%	77.04	79.74	67.75	68.97	76.34	74.28	81.26	81.52	61.70	60.00	72.86
80% + 20%	84.10	86.06	74.26	76.17	79.90	78.03	83.08	83.52	69.70	66.50	78.13
90% + 10%	90.88	92.24	80.79	83.18	83.51	81.95	84.46	85.20	78.20	73.70	83.41
100% + 0%	97.91	98.90	87.29	90.29	87.16	85.79	86.06	86.94	86.70	81.00	88.80
Average	63.21	67.39	54.88	55.16	69.10	66.37	77.32	77.48	43.83	44.40	61.91

Table 8.4 lists the average accuracy of models on test sets under different distributions. We can see that the accuracy degrades gradually when the test set includes OOD data, which confirms that distribution shift indeed weakens the reliability of the pre-trained DNN and it is necessary to enhance this DNN.

8.3.5 Retraining Settings

Like previous studies [178, 177, 95] and following our working assumptions, during the retraining process, the hyperparameters are set in the same way as the pre-trained DNN, such as the DNN architecture, momentum, batch size, activation function, dropout, optimization, learning rate, and loss function. Besides, for the number of epochs, we retrain the LeNet-1 and LeNet-5 with additional 5 epochs as [178], 10 epochs for ResNet-20 and NiN as [196]. We do not follow the same setting as [178] to use 5 epochs to retrain the CIFAR-10 based models. The reason is that we found in some cases, 5 epochs are not enough for the model weights to converge. As shown in Fig 8.3, for MNIST-based models, the test accuracy of original test data and new test data are almost the same after using 5, 10, and 15 epochs to retrain the models. However, for the CIFAR-10 based models, there are clear gaps of the test accuracy on the new data when using 5 epochs to retrain models compared with using 10 and 15 epochs to retrain, especially when the labeling budgets are 3% and 5%. Since this is the first work to evaluate the aforementioned selection metrics for model retraining on text datasets, we follow our practical experience to set 5 epochs to retrain IMDb- and Newsgroups-based models.



(a) MNIST, LeNet-5 (b) CIFAR-10, ResNet20

Figure 8.3: Test accuracy of original test data and new test data after using random selection metric to select different budgets of data and retrain the model. 5-ori means the test accuracy on original test data after retraining the model with 5 epochs.

8.3.6 Repetitions and Infrastructure

Each experiment is repeated 5 times to reduce the randomness introduced in the training process. All the experiments run on a high-performance computer cluster, and each cluster node runs a 2.6 GHz Intel Xeon Gold 6132 CPU with an NVIDIA Tesla V100 16G SXM2 GPU.

8.4 Experimental Results

We report the experimental results to answer each RQ and summarize our findings. Remember that the combined candidate set without labels represents the new coming data where selected data for model retraining come from. The combined test set with labels is for testing the resulting accuracy of DNNs. We create these two sets in a way that they contain the same percentage of ID data and OOD data.

8.4.1 RQ1: Different Retraining processes

Our goal is to analyze which retraining process can maintain high accuracy on original test data and meanwhile achieve high accuracy on new data. We denote by **Type 1** the process that retrains the model with the new data only, and by **Type 2** the process that retrains the model using a combination of new data and previous training data. To determine which retraining process is better, we compare the accuracy improvement of DNNs after retraining using each process. For each DNN, we create 11 sets of unlabeled data as well as 11 sets of test data following different data distributions by combining ID and OOD data. Next, each metric (of 6) selects a certain ratio (budget) of data from each unlabeled candidate set for labeling and model retraining. In our study, the ratio of selected data is set to 1%, 3%, 5%, and 10% as [178]. Besides, to exclude the effect of selection metrics on the retrained models, we also consider using all the candidate data (i.e., with budget 100%) to retrain the DNN models by different retraining processes. Finally, we calculate the accuracy improvement of DNNs after retraining. In total, we have retrained 71280 DNN models, 3 datasets $\times$ 2 models $\times$ (6 selection metrics $\times$ 4 budgets + 1 budget) $\times$ 11 distributions $\times$ 8 operators $\times$ 5 repetitions image-based models, and 2 datasets $\times$ 2 models $\times$ 6 selection metrics $\times$ 4 budgets $\times$ 11 distributions $\times$ 5 repetitions text-based models. Table 8.5 and Table 8.6 show the statistical improvements of test

accuracy over the 71280 DNNs of the original and new test data, respectively. In each table, the first column represents the data distribution of the candidate set. For instance, 10% + 90% indicates that the candidate set consists of 10% ID data and 90% OOD data.

In the case of maintaining performance on the original test set, as demonstrated by Table 8.5, in most cases (512 out of 550) over 5 datasets, the retraining process of **Type 2** achieves better results than **Type 1**. And on average, in all the cases, the retraining process of **Type 2** achieves better (by up to 29.52%) results than **Type 1**. Namely, retraining using the combination of newly selected data and training data is a better option than using only the newly selected data for this objective. Now look into Table 8.6, surprisingly, retraining with only the new data does not ensure higher accuracy on the new test data in most cases. In general, only in 153 cases (out of 550 cases), the retraining process of **Type 1** achieves better accuracy than **Type 2**. On average, we can see that only when more (at least 80%) OOD data are included in the candidate set, the retraining process of **Type 1** can achieve better results (by up to 4.28%) than **Type 2**. Note that, meanwhile, the accuracy of the original test data is greatly sacrificed. For instance, in the case of 100% OOD data and **Budget 10%**, **Type 1** improves the accuracy on the new test set by 48.46%, but the accuracy on the original test set drops significantly by 29.84%. Besides, this outperformance on new test sets degrades with a smaller budget is available. Overall, on average, retraining using both the training data and the newly selected data better enhances the model without losing the high performance on the original test data. We can conclude that this retraining strategy achieves a good balance between the original and new test sets.

Answer to RQ1: Retraining DNNs with only the newly selected data sacrifices accuracy on the original distribution for improvement on the new distribution. By contrast, mixing the training data with the new data can achieve high accuracy on both the original and the new distributions. More specifically, retraining on new data achieves higher test accuracy only when there are more than 80% OOD data in the candidate set. Overall, combining training data and selected data remains the best option.

8.4.2 RQ2: Effectiveness of Different selection metrics

Based on the answer of **RQ1**, we use the retraining process that combines the training and new data for our remaining studies. Besides, since the accuracy of the original test set is highly maintained, we only consider the performance of the new test set in the following RQs.

We observe that the evaluation of existing selection metrics for model retraining lacks insights regarding the amplitude of the distribution shift. For example, MCP is evaluated by only using one data combination (80% original test data + 20% mutated data), and DeepGini is evaluated by only (100%) mutated data. Thus, the actual effectiveness of these metrics when facing different data distributions is ambiguous. In this research question, we explore how different distributions of candidate data affect the effectiveness of each metric for model enhancement. To achieve this, we still follow the same experimental setting as our first study. In total, for each image dataset, each test combination has 64 (2 models × 8 operators × 4 budgets) retraining performances averaging on 5 times repetitions. In this section, we only report the results of image datasets due that we use natural OOD data

Table 8.5: Average (over all selection metrics) improvement of test accuracy (%) on **original** test sets (ID test data) with different selection budgets. The better result between the two types of retraining processes is highlighted in gray. **Type 1**: using only the new data; **Type 2**: using the combination of newly selected data and training data. "**Distribution**" represents different distribution shifts by different percentages of ID and OOD data in the candidate set. Baseline: Please refer to Table 8.1 for the accuracy of pre-trained DNNs.

	Distribution	Budget 1%		Budget 3%		Budget 5%		Budget 10%		Budget 100%	
	ID + OOD	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2
MNIST LeNet1	0% + 100%	-8.50	0.36	-18.87	0.32	-16.82	0.27	-22.28	0.25	-29.04	-0.06
	10% + 90%	-5.25	0.40	-15.35	0.35	-14.64	0.34	-21.11	0.27	-8.27	-0.02
	20% + 80%	-4.70	0.40	-11.79	0.34	-9.96	0.31	-21.46	0.27	-5.35	-0.01
	30% + 70%	-3.71	0.39	-9.30	0.35	-9.08	0.35	-17.07	0.28	-3.92	0.04
	40% + 60%	-2.39	0.39	-9.68	0.37	-9.08	0.37	-15.11	0.26	-2.91	0.08
	50% + 50%	-1.07	0.41	-5.54	0.38	-6.99	0.38	-13.08	0.28	-2.27	0.15
	60% + 40%	-0.92	0.40	-4.97	0.38	-4.58	0.37	-7.82	0.31	-1.82	0.20
	70% + 30%	-0.65	0.39	-3.14	0.40	-3.91	0.37	-6.76	0.34	-1.41	0.20
	80% + 20%	-0.89	0.42	-2.62	0.38	-2.42	0.40	-3.05	0.35	-1.07	0.25
	90% + 10%	-0.36	0.42	-1.44	0.41	-1.22	0.38	-1.36	0.40	-0.66	0.30
	100% + 0%	0.07	0.42	0.07	0.44	0.14	0.47	0.19	0.47	0.16	0.47
	Average	-2.58	0.40	-7.51	0.37	-7.14	0.37	-11.72	0.32	-5.14	0.14
F-MNIST LeNet1	0% + 100%	-20.65	0.07	-23.08	-0.02	-24.28	-0.35	-23.59	-0.15	-28.63	-1.05
	10% + 90%	-13.23	0.19	-16.27	0.07	-17.83	-0.26	-17.57	-0.14	-9.24	-0.92
	20% + 80%	-9.80	0.17	-13.47	0.01	-14.81	-0.24	-15.03	-0.07	-6.81	-0.77
	30% + 70%	-6.05	0.04	-11.65	0.05	-13.12	-0.07	-13.26	0.03	-5.45	-0.72
	40% + 60%	-3.60	0.08	-10.37	0.12	-10.72	0.01	-11.72	0.03	-4.45	-0.49
	50% + 50%	-3.31	0.14	-9.43	0.12	-10.02	0.00	-10.25	0.04	-3.85	-0.47
	60% + 40%	-2.51	0.11	-5.56	0.22	-8.26	0.01	-9.48	0.05	-3.18	-0.32
	70% + 30%	-1.97	0.12	-3.99	0.17	-5.94	-0.04	-6.88	0.10	-2.48	-0.21
	80% + 20%	-1.84	0.03	-2.45	0.20	-4.01	0.01	-5.21	0.18	-1.89	-0.13
	90% + 10%	-1.07	0.04	-1.55	0.24	-2.38	0.16	-2.70	0.19	-1.19	0.04
	100% + 0%	-0.40	-0.03	-0.57	0.30	-0.52	0.10	-0.25	0.29	0.01	0.27
	Average	-5.86	0.09	-8.94	0.14	-10.17	-0.06	-10.54	0.05	-6.10	-0.43
CIFAR-10 ReNet20	0% + 100%	-6.64	-0.43	-9.39	-0.61	-9.31	-0.52	-11.37	-0.59	-8.08	-0.39
	10% + 90%	-3.74	-0.59	-7.97	-0.37	-6.84	-0.54	-9.94	-0.58	-3.70	-0.31
	20% + 80%	-2.10	-0.52	-6.19	-0.38	-5.92	-0.39	-8.66	-0.44	-2.97	-0.34
	30% + 70%	-2.13	-0.46	-6.44	-0.35	-4.50	-0.40	-7.38	-0.26	-2.31	0.00
	40% + 60%	-1.33	-0.57	-3.54	-0.32	-2.98	-0.24	-5.92	-0.23	-1.82	-0.21
	50% + 50%	-0.37	-0.38	0.11	-0.28	-2.67	-0.24	-6.14	-0.38	-1.58	-0.03
	60% + 40%	0.88	-0.35	0.61	-0.30	-0.04	-0.20	-4.69	-0.18	-1.26	0.08
	70% + 30%	1.06	-0.30	0.55	-0.25	-0.09	-0.21	-4.61	-0.16	-0.77	0.09
	80% + 20%	1.18	-0.24	0.51	-0.17	0.29	-0.19	-3.43	-0.14	-0.43	0.15
	90% + 10%	1.18	-0.19	0.58	-0.19	0.26	-0.18	-2.97	-0.13	0.12	0.18
	100% + 0%	1.14	-0.24	0.57	-0.21	0.66	-0.13	-4.35	-0.14	0.60	0.27
	Average	-0.99	-0.39	-2.78	-0.31	-2.83	-0.29	-6.31	-0.29	-2.02	-0.05
IMDB LSTM	0% + 100%	-0.44	0.68	-2.20	0.78	-3.76	0.75	-2.13	0.71	-6.07	0.78
	10% + 90%	-0.16	0.64	-1.27	0.70	-1.12	0.69	-0.68	0.71	-0.31	0.68
	20% + 80%	-0.25	0.49	-1.54	0.82	-1.29	0.81	0.09	0.63	1.23	1.01
	30% + 70%	-0.30	0.75	-1.32	0.78	-0.10	0.77	0.35	0.78	0.81	1.07
	40% + 60%	-0.49	0.81	-0.03	0.86	0.38	0.85	0.33	0.87	1.57	1.16
	50% + 50%	0.27	0.90	-0.21	0.85	0.39	0.85	0.67	0.87	1.75	1.31
	60% + 40%	0.36	0.93	-0.45	0.86	0.49	0.86	0.58	0.97	1.73	1.30
	70% + 30%	0.27	0.87	-0.34	0.86	0.08	0.95	0.79	1.01	1.92	1.60
	80% + 20%	0.18	0.85	0.48	0.90	0.68	0.90	0.96	0.95	1.77	1.47
	90% + 10%	0.35	0.89	0.18	0.80	0.45	1.04	1.14	1.00	2.04	1.58
	100% + 0%	0.17	0.85	0.63	0.98	0.74	0.94	0.89	1.03	2.01	1.65
	Average	0.00	0.79	-0.55	0.83	-0.28	0.86	0.27	0.87	0.77	1.24
Newsgroups NN	0% + 100%	-27.82	0.33	-62.95	0.46	-66.43	0.39	-67.55	0.19	-79.23	-2.37
	10% + 90%	-18.86	0.43	-47.79	0.29	-45.39	0.40	-37.10	0.50	-29.82	-1.37
	20% + 80%	-14.97	0.30	-32.33	0.51	-31.81	0.40	-25.16	0.38	-15.26	-0.54
	30% + 70%	-12.22	0.31	-16.04	0.48	-20.89	0.45	-15.80	0.40	-7.11	-0.08
	40% + 60%	-9.20	0.54	-7.64	0.44	-15.92	0.52	-10.75	0.57	-4.53	0.11
	50% + 50%	-2.75	0.57	-6.31	0.56	-11.86	0.58	-7.54	0.50	-2.20	0.75
	60% + 40%	-1.72	0.42	-3.69	0.60	-6.30	0.60	-3.81	0.69	-0.31	1.37
	70% + 30%	-0.16	0.66	-3.23	0.68	-2.68	0.81	-1.69	0.75	0.41	1.84
	80% + 20%	0.14	0.79	-0.69	0.79	-0.53	0.72	-0.55	0.96	1.36	2.17
	90% + 10%	0.21	0.60	0.01	0.65	-0.14	0.89	0.18	1.11	1.73	2.22
	100% + 0%	0.20	0.62	-0.01	0.68	0.57	0.84	0.66	1.14	2.22	2.59
	Average	-7.92	0.51	-16.42	0.56	-18.31	0.60	-15.37	0.65	-12.07	0.61
MNIST LeNet5	0% + 100%	-13.41	-0.03	-25.70	-0.03	-25.95	-0.04	-35.98	-0.01	-30.84	0.01
	10% + 90%	-12.39	-0.02	-22.47	-0.04	-23.83	-0.03	-28.27	-0.02	-4.21	0.02
	20% + 80%	-10.37	-0.03	-19.42	0.00	-18.54	0.01	-23.72	-0.02	-2.61	0.02
	30% + 70%	-8.14	-0.04	-17.83	-0.06	-15.01	-0.03	-18.59	-0.01	-1.78	0.05
	40% + 60%	-7.70	-0.04	-12.87	-0.04	-11.73	-0.04	-13.30	-0.01	-1.37	0.02
	50% + 50%	-6.70	-0.03	-9.99	-0.01	-8.53	0.01	-9.84	0.01	-1.05	0.04
	60% + 40%	-4.11	-0.05	-6.40	-0.03	-5.53	-0.01	-9.51	-0.01	-0.73	0.05
	70% + 30%	-3.32	-0.03	-4.91	0.00	-4.72	0.02	-6.09	0.02	-0.53	0.05
	80% + 20%	-1.84	0.01	-3.48	-0.01	-3.22	0.01	-4.40	0.03	-0.38	0.04
	90% + 10%	-0.96	0.03	-2.62	0.04	-1.66	0.04	-2.23	0.06	-0.27	0.05
	100% + 0%	-0.75	0.06	-0.69	0.08	-0.18	0.07	-0.20	0.07	-0.07	0.06
	Average	-6.34	-0.02	-11.49	-0.01	-10.81	0.00	-13.83	0.01	-3.99	0.04
F-MNIST LeNet5	0% + 100%	-19.26	0.07	-22.65	0.00	-20.98	-0.22	-20.46	-0.07	-25.62	-0.24
	10% + 90%	-13.63	0.00	-19.67	0.05	-18.48	-0.24	-18.42	-0.05	-5.90	-0.22
	20% + 80%	-9.83	0.04	-16.87	0.05	-17.59	-0.06	-17.33	0.00	-4.35	-0.11
	30% + 70%	-8.23	-0.01	-14.81	0.05	-17.23	-0.09	-15.42	-0.05	-3.14	-0.04
	40% + 60%	-6.97	0.04	-13.98	0.00	-15.21	-0.11	-14.14	-0.04	-2.36	0.07
	50% + 50%	-6.51	0.02	-11.44	0.02	-13.79	-0.03	-13.91	0.04	-1.89	0.08
	60% + 40%	-4.03	0.09	-9.71	0.14	-12.37	-0.18	-11.77	0.10	-1.57	0.16
	70% + 30%	-2.70	0.10	-9.22	0.15	-10.84	0.05	-10.12	0.19	-1.28	0.21
	80% + 20%	-1.91	0.23	-6.31	0.16	-8.84	0.08	-7.62	0.18	-0.93	0.26
	90% + 10%	-1.33	0.24	-4.44	0.17	-5.96	0.07	-4.37	0.24	-0.59	0.35
	100% + 0%	0.00	0.23	0.11	0.25	-0.02	0.15	-0.45	0.35	0.12	0.44
	Average	-6.76	0.10	-11.73	0.09	-12.85	-0.05	-12.18	0.08	-4.32	0.09
CIFAR-10 NiN	0% + 100%	-14.28	-0.58	-16.17	-0.70	-18.82	-0.57	-22.54	-0.63	-13.47	-0.45
	10% + 90%	-12.13	-0.56	-17.90	-0.51	-15.82	-0.58	-18.16	-0.47	-6.87	-0.41
	20% + 80%	-7.56	-0.51	-12.84	-0.54	-13.40	-0.45	-16.12	-0.43	-5.77	-0.29
	30% + 70%	-3.31	-0.48	-9.54	-0.36	-11.45	-0.35	-16.14	-0.28	-5.14	-0.24
	40% + 60%	-2.34	-0.46	-3.92	-0.40	-8.29	-0.35	-14.24	-0.23	-4.75	-0.30
	50% + 50%	-1.81	-0.41	-3.05	-0.33	-4.72	-0.34	-12.74	-0.29	-4.28	-0.18
	60% + 40%	-1.78	-0.41	-2.82	-0.28	-3.64	-0.33	-10.09	-0.29	-3.83	-0.21
	70% + 30%	-1.66	-0.39	-2.95	-0.31	-3.78	-0.37	-9.59	-0.19	-3.19	0.00
	80% + 20%	-1.63	-0.36	-2.98	-0.32	-3.51	-0.25	-8.20	-0.15	-2.67	-0.02
	90% + 10%	-1.61	-0.35	-2.72	-0.25	-3.30	-0.24	-7.59	-0.21	-2.29	0.13
	100% + 0%	-1.64	-0.30	-2.48	-0.36	-3.40	-0.23	-7.20	-0.18	-1.97	0.21
	Average	-4.52	-0.44	-7.04	-0.40	-8.19	-0.37	-12.96	-0.30	-4.93	-0.16
IMDB GRU	0% + 100%	-0.03	0.54	-1.78	0.42	-4.42	0.34	-4.36	0.46	-4.78	0.59
	10% + 90%	-0.12	0.47	-2.34	0.34	-1.30	0.48	-1.88	0.47	0.36	0.67
	20% + 80%	-0.06	0.51	-0.57	0.56	-1.68	0.37	-0.52	0.42	0.69	1.01
	30% + 70%	-0.18	0.45	-0.42	0.47	-0.52	0.56	-0.32	0.52	0.35	1.03
	40% + 60%	-0.31	0.49	-0.18	0.54	-0.02	0.47	0.12	0.48	1.19	0.88
	50% + 50%	0.16	0.41	-0.08	0.63	0.18	0.64	0.46	0.60	1.05	1.25
	60% + 40%	0.13	0.61	0.05	0.47	0.43	0.67	0.43	0.73	1.81	1.40
	70% + 30%	0.11	0.49	0.25	0.52	0.19	0.64	0.74	0.67	1.97	1.49
	80% + 20%	0.09	0.64	0.18	0.51	0.24	0.67	0.64	0.76	1.95	1.52
	90% + 10%	0.15	0.57	0.16	0.66	0.39	0.58	0.84	0.75	2.06	1.56
	100% + 0%	0.16	0.52	0.10	0.65	0.63	0.67	0.85	0.78	1.95	1.71
	Average	0.01	0.52	-0.42	0.52	-0.53	0.55	-0.27	0.60	0.78	1.19
Newsgroups NN2	0% + 100%	-73.60	0.75	-72.67	0.16	-67.37	0.36	-63.77	-0.31	-72.62	-0.03
	10% + 90%	-48.54	1.27	-32.12	0.59	-29.14	0.44	-27.85	0.31	-16.06	1.83
	20% + 80%	-38.16	1.18	-25.26	0.88	-18.47	0.27	-20.16	0.78	-5.90	2.40
	30% + 70%	-25.13	1.11	-18.88	1.04	-15.75	1.16	-13.60	0.80	-2.32	3.10
	40% + 60%	-18.16	1.19	-16.44	0.85	-13.18	1.19	-9.46	1.52	-0.39	3.01
	50% + 50%	-12.45	1.73	-10.30	1.62	-9.17	1.40	-8.39	1.94	1.83	4.14
	60% + 40%	-9.84	1.52	-8.13	1.65	-7.01	2.26	-6.43	2.04	3.18	4.77
	70% + 30%	-8.34	1.34	-6.69	1.82	-4.93	2.05	-3.83	2.48	3.88	5.16
	80% + 20%	-7.94	1.56	-4.04	1.90	-3.24	2.02	-1.81	2.64	4.65	5.71
	90% + 10%	-2.83	1.76	-4.30	2.18	-1.46	2.58	0.78	2.96	5.19	6.14
	100% + 0%	-5.62	1.81	-1.45	2.17	-0.12	2.74	1.90	3.45	6.19	6.30
	Average	-22.78	1.38	-18.21	1.35	-15.44	1.50	-13.87	1.69	-6.58	3.86

	Distribution	Budget 1%		Budget 3%		Budget 5%		Budget 10%		Budget 100%	
	ID + OOD	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2
Average	0% + 100%	-18.46	0.18	-25.55	0.08	-25.81	0.04	-27.40	-0.01	-29.84	-0.32
	10% + 90%	-12.80	0.22	-18.32	0.15	-17.44	0.07	-18.10	0.10	-8.40	-0.01
	20% + 80%	-9.78	0.20	-14.03	0.22	-13.35	0.10	-14.81	0.15	-4.71	0.24
	30% + 70%	-6.94	0.21	-10.02	0.24	-10.76	0.23	-11.72	0.22	-3.00	0.42
	40% + 60%	-5.25	0.25	-7.86	0.24	-8.68	0.27	-9.42	0.32	-1.98	0.43
	50% + 50%	-3.46	0.34	-5.62	0.36	-6.72	0.33	-8.07	0.36	-1.25	0.70
	60% + 40%	-2.35	0.33	-4.11	0.37	-4.68	0.41	-6.26	0.44	-0.60	0.88
	70% + 30%	-1.74	0.32	-3.37	0.40	-3.66	0.43	-4.80	0.52	-0.15	1.04
	80% + 20%	-1.44	0.39	-2.14	0.43	-2.46	0.44	-3.27	0.58	0.24	1.14
	90% + 10%	-0.63	0.40	-1.62	0.47	-1.50	0.53	-1.83	0.64	0.61	1.26
	100% + 0%	-0.67	0.39	-0.37	0.50	-0.15	0.56	-0.80	0.73	1.12	1.40
	Average	-5.77	0.29	-8.51	0.32	-8.66	0.31	-9.68	0.37	-4.36	0.65

Table 8.6: Average (over all selection metrics) improvement of test accuracy (%) on **new** test sets with different selection budgets. The better result between the two types of retraining processes is highlighted in gray. **Type 1**: using only the new data; **Type 2**: using the combination of new selected data and training data. "**Distribution**" represents different distribution shifts of the candidate set. Baseline: Please refer to Table 8.4 for the accuracy of pre-trained DNNs.

Distribution		Budget 1%		Budget 3%		Budget 5%		Budget 10%		Budget 100%	
ID + OOD		Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2
MNIST LeNet1	0% + 100%	6.71	9.86	15.22	16.73	19.23	21.43	35.63	30.88	61.87	63.82
	10% + 90%	5.79	9.20	12.49	14.84	15.84	18.57	29.51	27.38	54.96	56.97
	20% + 80%	4.99	8.20	9.41	13.02	13.24	16.50	24.31	24.04	48.24	50.11
	30% + 70%	4.41	7.47	8.20	11.57	11.15	14.48	18.63	20.24	41.50	43.33
	40% + 60%	3.79	6.70	5.97	9.95	8.58	12.64	14.06	17.64	34.92	36.56
	50% + 50%	2.87	5.61	3.89	8.42	6.29	11.00	9.93	14.92	28.46	29.89
	60% + 40%	2.44	4.62	3.23	7.00	4.28	9.03	6.65	12.22	22.01	23.24
	70% + 30%	1.36	3.40	1.82	5.46	2.53	6.95	3.85	9.30	15.82	16.58
	80% + 20%	0.70	2.28	0.53	3.85	1.30	4.63	2.69	6.20	9.54	10.09
	90% + 10%	0.32	1.32	0.14	2.01	0.66	2.42	1.68	3.09	3.87	4.20
	100% + 0%	0.07	0.41	0.07	0.43	0.14	0.47	0.18	0.47	0.15	0.46
	Average	3.04	5.37	5.54	8.48	7.57	10.74	13.38	15.13	29.21	30.48
F-MNIST LeNet1	0% + 100%	10.15	4.59	19.07	8.02	26.35	12.59	32.24	20.28	54.49	51.27
	10% + 90%	4.11	4.28	13.73	6.79	21.94	9.67	28.71	16.44	47.60	45.19
	20% + 80%	3.21	3.81	8.59	5.96	15.37	8.56	22.37	13.21	41.54	39.82
	30% + 70%	2.68	3.78	6.45	5.46	10.70	7.52	15.81	11.83	35.43	34.12
	40% + 60%	1.51	3.12	4.82	4.66	8.39	6.76	12.07	10.09	29.68	28.58
	50% + 50%	0.57	2.60	2.73	4.11	5.05	5.74	8.28	8.53	23.54	22.90
	60% + 40%	-0.36	1.98	1.65	3.20	3.20	4.60	5.07	6.87	18.01	17.60
	70% + 30%	-0.31	1.52	1.50	2.64	1.48	3.39	2.86	5.09	12.92	12.53
	80% + 20%	-0.71	1.12	0.84	1.98	-0.20	2.32	0.57	3.49	7.55	7.41
	90% + 10%	-0.66	0.68	-0.32	1.14	-0.98	1.24	-0.63	1.71	2.83	2.92
	100% + 0%	-0.40	0.26	-0.56	0.31	-0.53	0.12	-0.26	0.30	0.02	0.27
	Average	1.80	2.52	5.32	4.02	8.25	5.68	11.55	8.90	24.87	23.87
CIFAR-10 ReNet20	0% + 100%	12.94	7.30	14.70	8.65	18.00	10.76	22.69	14.49	33.81	27.26
	10% + 90%	7.41	6.27	9.63	7.21	13.33	8.58	18.37	11.84	29.76	23.95
	20% + 80%	6.56	5.74	7.82	5.99	9.31	7.21	12.66	9.39	26.08	20.95
	30% + 70%	5.49	4.96	5.85	5.52	7.27	6.14	8.66	7.74	22.37	17.83
	40% + 60%	4.53	4.13	4.55	4.20	5.74	4.84	6.28	5.59	18.62	14.55
	50% + 50%	3.59	3.44	3.67	3.49	3.94	3.96	4.06	4.43	15.10	11.58
	60% + 40%	2.81	0.40	2.96	2.47	2.76	2.99	2.24	3.21	11.46	8.81
	70% + 30%	2.33	-0.08	2.12	1.75	1.85	1.93	0.08	1.93	7.95	6.03
	80% + 20%	1.85	-0.64	1.52	0.98	1.34	1.17	-1.31	1.09	4.91	3.68
	90% + 10%	1.46	-1.11	1.01	0.20	0.67	0.34	-2.45	0.20	2.33	1.53
	100% + 0%	1.11	-1.46	0.55	-0.22	0.63	-0.13	-4.34	-0.48	0.60	0.27
	Average	4.55	2.63	4.94	3.66	5.90	4.34	6.08	5.40	15.73	12.40
IMDB LSTM	0% + 100%	0.83	0.42	0.88	0.65	1.17	0.55	1.28	0.63	2.63	1.89
	10% + 90%	0.48	0.57	0.50	0.71	0.92	0.46	0.93	0.59	1.90	2.01
	20% + 80%	0.36	0.57	0.44	0.72	0.80	0.66	1.14	0.58	1.87	1.59
	30% + 70%	0.26	0.51	0.07	0.54	0.71	0.61	0.53	0.54	1.83	1.67
	40% + 60%	0.01	0.48	0.37	0.55	0.27	0.33	0.85	0.41	1.67	1.51
	50% + 50%	0.33	0.60	0.17	0.51	0.64	0.53	0.94	0.63	2.03	1.14
	60% + 40%	0.34	0.55	-0.30	0.74	0.21	0.59	0.94	0.74	1.70	1.45
	70% + 30%	0.17	0.33	-0.39	0.66	0.11	0.53	1.06	0.65	2.14	1.21
	80% + 20%	0.11	0.59	0.31	0.67	0.58	0.65	1.03	0.61	1.81	1.18
	90% + 10%	0.32	0.80	0.17	0.75	0.48	0.95	1.15	0.96	2.02	1.43
	100% + 0%	0.17	0.85	0.63	0.98	0.74	0.94	0.89	1.03	2.01	1.65
	Average	0.31	0.57	0.26	0.68	0.60	0.62	0.98	0.67	1.97	1.52
Newsgroups NN	0% + 100%	18.78	7.65	30.21	17.30	37.25	26.71	48.27	44.86	94.37	93.90
	10% + 90%	14.13	6.20	21.24	14.69	26.63	22.30	38.04	38.34	81.79	83.70
	20% + 80%	12.52	5.24	14.22	12.40	21.84	19.64	31.35	32.65	72.48	73.93
	30% + 70%	9.28	4.97	9.91	9.33	16.12	15.57	25.12	26.33	63.41	64.06
	40% + 60%	6.41	3.93	7.62	7.79	9.67	11.87	20.53	21.43	52.36	53.43
	50% + 50%	3.13	2.92	4.26	5.42	5.83	8.48	14.03	15.84	41.83	42.38
	60% + 40%	2.56	2.25	2.98	4.25	5.09	5.82	10.33	11.10	31.67	31.86
	70% + 30%	1.27	1.64	1.55	3.15	3.69	4.38	6.53	7.05	21.60	22.35
	80% + 20%	0.95	1.60	0.95	1.94	2.15	2.42	2.95	3.76	13.16	13.07
	90% + 10%	0.53	0.91	0.68	1.15	1.06	1.79	1.17	2.01	5.24	5.67
	100% + 0%	0.20	0.62	-0.01	0.68	0.57	0.84	0.66	1.14	2.22	2.59
	Average	6.34	3.45	8.51	7.10	11.81	10.89	18.09	18.59	43.65	44.27
NewsgroupsNN2	0% + 100%	24.99	10.82	34.88	23.09	45.11	32.40	55.98	46.03	86.88	85.08
	10% + 90%	15.89	8.42	26.31	20.39	33.01	26.86	44.90	39.01	75.94	76.00
	20% + 80%	9.18	6.60	18.75	16.92	25.30	23.10	36.39	33.74	66.53	67.30
	30% + 70%	4.85	5.47	12.70	13.49	19.80	18.53	29.51	28.41	57.79	58.76
	40% + 60%	0.47	3.99	6.67	10.48	12.93	14.74	21.26	21.65	48.04	48.83
	50% + 50%	-2.20	3.15	3.29	7.72	7.32	11.24	15.80	16.87	39.35	39.73
	60% + 40%	-1.73	2.60	1.36	5.89	4.63	8.03	10.56	12.43	31.13	31.06
	70% + 30%	-3.11	2.00	0.40	4.64	2.55	6.71	7.16	9.28	23.29	23.71
	80% + 20%	-4.58	1.84	-1.20	3.17	0.52	4.36	3.92	6.09	15.85	16.00
	90% + 10%	-2.13	1.89	-3.23	2.67	-0.25	3.20	2.49	4.20	9.23	9.89
	100% + 0%	-5.62	1.81	-1.45	2.17	-0.12	2.74	1.90	3.45	6.19	6.30
	Average	3.27	4.42	8.95	10.06	13.71	13.81	20.90	20.11	41.84	42.06

Distribution		Budget 1%		Budget 3%		Budget 5%		Budget 10%		Budget 100%	
ID + OOD		Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2
Average	0% + 100%	11.94	8.39	18.02	13.74	22.81	18.41	29.68	25.89	48.46	47.77
	10% + 90%	8.14	7.40	13.82	12.07	17.98	15.54	24.44	22.21	42.61	42.62
	20% + 80%	6.15	6.44	9.90	10.45	13.75	13.64	19.56	19.10	37.44	37.51
	30% + 70%	4.37	5.79	7.07	8.94	10.55	11.55	14.93	16.20	32.42	32.53
	40% + 60%	2.90	4.55	5.00	7.53	7.42	9.71	11.31	13.45	27.22	27.30
	50% + 50%	1.79	3.71	3.16	6.18	4.74	7.98	7.86	10.94	22.17	22.14
	60% + 40%	1.20	2.72	2.10	4.96	3.17	6.19	5.11	8.50	17.15	17.20
	70% + 30%	0.48	1.88	0.96	3.80	1.70	4.70	2.89	6.20	12.47	12.53
	80% + 20%	-0.07	1.24	0.14	2.57	0.51	3.07	0.97	3.99	7.86	7.94
	90% + 10%	-0.13	0.63	-0.62	1.48	-0.21	1.76	-0.16	2.14	3.73	3.92
	100% + 0%	-0.70	0.02	-0.37	0.50	-0.15	0.57	-0.80	0.64	1.13	1.40
	Average	3.28	3.89	5.38	6.57	7.48	8.46	10.53	11.75	22.97	22.99

for text datasets, whereas we can experimentally control the OOD data produced for images. Therefore, there are only a few combinations (8) for text data, and the statistical results are insufficient to draw conclusions. We report the results of text datasets in Section 8.5.4 (where we consider real-world distribution shift) by using the test accuracy improvement after retraining as the measurement metric.

Table 8.7 lists the frequency of each selection metric achieving the top-1 and top-3 best test accuracy over the 64 cases in each test combination. Note that, we also report top-3 results since if the metric achieves top-3 best performance, it outperforms half of the metrics. Interestingly, when the new set contains more than 70% OOD data, random selection defeats the other five, carefully-designed metrics in most cases (20 out of 24). Moreover, in total, the frequency of random selection being the best is almost twice the second-best metric. For example, the random selection obtains 90 times the top-1 best performance in the 100% OOD test set, while the second-best, CES, only reaches 47 times. Besides, when the included OOD data are more than 70%, the two metrics CES and DSA outperform the uncertain-based metrics, Entropy, DeepGini, and MCP. The reason is that when the new data consists of too much OOD data, a massive amount of information related to the new distribution has not been learned by the pre-trained model. In this case, the model needs to learn more from a representative sample of the new data rather than from the most uncertain data. Among all the studied metrics, random selection is the most effective because it does not bias the selection towards specific data and, therefore, achieves better representativeness.

With the increase of ID data in the candidate set and test set, the uncertain-based metrics, Entropy, DeepGini, and MCP, achieve better results than the other 3 metrics, CES, DSA, and random selection. More specifically, in total, when the proportion of OOD data is between 40% and 70%, MCP performs consistently better than all the others. On the other hand, when the test set contains more ($\geq$80%) ID data, Entropy and DeepGini achieve the best results. This is because as a higher ratio of ID data is part of the new distribution, the pre-trained model has already learned more from this new distribution. The OOD data, in this case, can be seen as outliers that generate uncertainty in the model and are, therefore, naturally selected by the uncertainty-based metrics. Hence, retraining on these data fills the gap in model learning and achieves better performance.

Answer to RQ2: None of the selection metrics outperforms the others across all ranges of distribution shifts. When the new set contains much more ($\geq$70%) OOD than ID data, the simple but effective random selection defeats the others. On the contrary, when the new set contains more ID data, the uncertain-based metrics are more effective.

8.4.3 RQ3: Distribution and Bias of Selected Data

Following our findings above, in this research question, we further explore another property that may impact the effectiveness of the selection metrics: class bias of the data selected by each metric. That is, we check if the selected data are evenly chosen from different classes, which is done by calculating the variance of labels of selected data. For example, given a 3 classes task, we select 100 data, if the number of selected data for each class is 30, 30, 40, the variance is $Variance(30, 30, 40) = 22.22$, while if the label numbers are 90, 5, 5, the variance should be $Variance(90, 5, 5) = 1605.55$. A small variance indicates a slight bias in data.

Table 8.7: Frequency of being the top-1 and top-3 best of the 6 selection metrics under different data distributions. The best result is highlighted in gray. "**Distribution**" represents different distribution shifts of the candidate set.

	Distribution ID + OOD	Entropy	DeepGini	MCP	CES	DSA	Random	Entropy	DeepGini	MCP	CES	DSA	Random
		Top-1						Top-3					
MNIST	0% + 100%	1	0	1	19	4	39	3	4	36	60	26	63
	10% + 90%	1	2	6	14	9	32	2	8	32	60	30	60
	20% + 80%	0	3	7	20	8	26	5	13	36	51	30	57
	30% + 70%	4	5	17	16	1	21	12	19	44	48	21	48
	40% + 60%	5	9	21	11	5	13	20	32	47	35	19	39
	50% + 50%	3	12	28	9	2	10	29	40	52	29	13	29
	60% + 40%	8	16	23	3	6	8	36	50	54	19	10	23
	70% + 30%	7	25	23	3	4	2	54	58	54	10	9	7
	80% + 20%	20	26	14	0	1	3	59	59	57	3	5	9
	90% + 10%	29	26	6	0	2	1	60	59	59	0	7	7
	100% + 0%	15	17	13	9	3	7	40	34	33	30	27	28
	Average	8.45	12.82	14.45	9.45	4.09	14.73	29.09	34.18	45.82	31.36	17.91	33.64
Fashion-MNIST	0% + 100%	0	2	3	13	11	35	3	3	26	60	40	60
	10% + 90%	1	0	6	14	8	35	2	3	30	60	42	55
	20% + 80%	0	0	4	12	15	33	1	5	32	58	40	56
	30% + 70%	1	2	8	12	16	25	6	4	41	53	35	53
	40% + 60%	0	0	14	14	19	17	6	9	38	49	42	48
	50% + 50%	1	4	16	9	20	14	14	17	39	36	36	50
	60% + 40%	6	9	23	3	16	7	21	31	43	25	40	32
	70% + 30%	6	14	30	2	9	3	31	40	52	20	27	22
	80% + 20%	15	16	22	3	5	3	44	42	56	13	21	16
	90% + 10%	22	16	18	2	4	2	50	53	58	9	14	8
	100% + 0%	16	15	9	10	9	5	35	36	37	32	26	26
	Average	6.18	7.09	13.91	8.55	12.00	16.27	19.36	22.09	41.09	37.73	33.00	38.73
CIFAR-10	0% + 100%	8	14	5	15	6	16	28	36	25	33	31	39
	10% + 90%	7	15	3	10	10	19	29	39	14	37	30	43
	20% + 80%	17	14	0	11	6	16	35	41	11	39	24	42
	30% + 70%	15	18	1	9	4	18	38	47	17	30	20	40
	40% + 60%	14	22	4	7	8	9	41	49	18	32	21	31
	50% + 50%	19	24	1	4	11	5	43	50	23	21	20	35
	60% + 40%	18	25	4	5	8	4	47	48	24	23	22	28
	70% + 30%	24	19	7	5	5	4	48	51	34	21	16	22
	80% + 20%	25	24	6	2	2	5	51	49	46	14	14	18
	90% + 10%	20	27	6	4	3	4	49	54	46	16	12	15
	100% + 0%	10	15	20	6	7	6	37	36	48	31	22	18
	Average	16.09	19.73	5.18	7.09	6.36	9.64	40.55	45.45	28.00	27.18	21.09	29.73
Total	0% + 100%	9	16	9	47	21	90	34	43	87	153	97	162
	10% + 90%	9	17	15	38	27	86	33	50	76	157	102	158
	20% + 80%	17	17	11	43	29	75	41	59	79	148	94	155
	30% + 70%	20	25	26	37	21	64	56	70	102	131	76	141
	40% + 60%	19	31	39	32	32	39	67	90	103	118	82	116
	50% + 50%	23	40	45	22	33	29	86	107	116	86	69	112
	60% + 40%	32	50	50	11	30	19	104	129	121	67	72	83
	70% + 30%	37	58	60	10	18	9	133	149	140	51	52	51
	80% + 20%	60	66	42	5	8	11	154	150	159	30	40	43
	90% + 10%	71	69	30	6	9	7	159	166	163	25	33	30
	100% + 0%	41	47	42	25	19	18	112	106	118	93	75	72
	Average	30.73	39.64	33.55	25.09	22.45	40.64	94.5	107.6	117.7	90.6	69.5	96.1

First, Figure 8.4 illustrates the data distribution of selected data by different metrics. Compared with the data distribution in the candidate set (black dashed line), three metrics, CES, DSA, and random, select ID and OOD data following almost the same distribution. On the contrary, the uncertain-based metrics, Entropy, DeepGini, and MCP, tend to pick more OOD than ID data. The reason is that the uncertain-based metrics always choose the most informative data, and the OOD data have likely not been learned by the pre-trained DNNs. Thus, there is more chance for OOD data to be selected by these uncertain-based metrics.

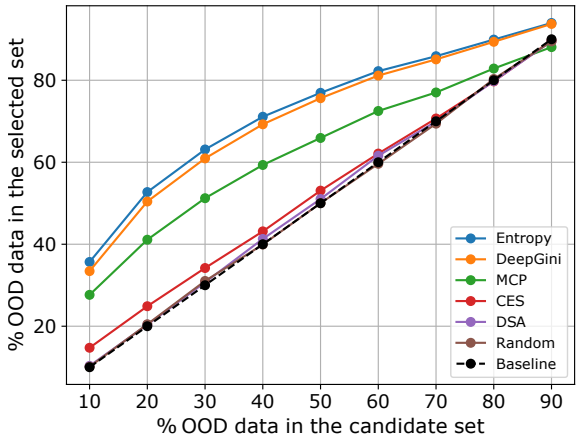


Figure 8.4: Comparison of data distributions of the selected set by different metrics. Baseline: The selected set has the same data distribution (same percentage of OOD and ID data) as the candidate set.

Second, Table 8.8 shows the class bias presented by the variance of labels of selected data. Compared with the other selection metrics, random always selects data evenly from different classes. However, the variances of two uncertainty-based metrics, Entropy and DeepGini, are more than twice the others. Although MCP is designed to select data evenly from different boundary areas, this metric has a higher bias than CES and random selection. The reason for Entropy, DeepGini, and MCP selecting bias classes is that they all use the predicted probability to measure the uncertainty, which is highly affected by the accuracy of the pre-trained model on the new data. When there are more OOD data, the prediction is more unreliable. For instance, MCP tends to decrease the bias in data when the proportion of ID data is above 50%.

Considering the results of RQ2, we conjecture that, when there are more OOD (e.g., $\geq$70%) data in the candidate set, it is better to select data with a better class balance to retrain the model. As an illustration of this hypothesis, random selection and CES achieve both higher accuracy and class balance. Since in the candidate set, most of the data have not been learned by the model, a better class balance can help represent a more diverse distribution and, in turn, lead the model to learn more diverse information.

Answer to RQ3: Uncertain-based selection metrics (Entropy, DeepGini, and MCP) tend to select more OOD data, and in a way that creates class imbalance in the set of retraining data. On the contrary, CES and random select data with more balanced classes and better representativeness of the new distribution. These two factors contribute to the difference in the effectiveness of the selection metrics, depending on how much ID data are still part of the new distribution.

Table 8.8: Class bias (label variance) of selected data by different metrics. The best result is highlighted in gray. "**Distribution**" represents different distribution shifts of the candidate set. The number means the average (over all selection metrics) variance in the number of examples that the metric selects for each class.

Distribution ID+OOD	Entropy	DeepGini	MCP	CES	DSA	Random
0% + 100%	479.42	429.86	361.13	213.13	279.09	188.75
10% + 90%	444.59	395.74	280.83	211.27	267.65	184.97
20% + 80%	423.30	377.19	270.06	210.26	263.01	182.18
30% + 70%	408.98	365.81	262.70	208.08	258.92	186.39
40% + 60%	387.65	350.30	250.64	207.57	256.99	181.09
50% + 50%	370.65	338.58	240.68	208.34	255.24	178.84
60% + 40%	350.82	324.47	233.23	209.83	255.07	177.36
70% + 30%	326.56	309.37	224.24	208.23	257.25	177.05
80% + 20%	307.41	299.57	218.07	207.98	261.64	178.10
90% + 10%	299.48	300.55	211.54	209.48	265.95	177.30
100% + 0%	392.50	385.06	216.62	213.89	268.97	177.93
Average	381.03	352.41	251.80	209.83	262.71	180.91

8.5 Distribution-aware Test Selection

According to the findings of our empirical study, when the new data contain more (≥60%) ID than OOD data, the uncertain-based metrics outperform others in enhancing the performance of DNNs. However, when there are more (≥70%) OOD data, none of the existing metrics (Entropy, DeepGini, MCP, CES, and DSA) defeats the random selection. There is, therefore, room for proposing a new metric to deal with the second case in a better way than random. **Intuition:** since different selection metrics behave differently on different data distributions, we should consider different selection strategies for different distributions of data. From the in-distribution data, we need to select the uncertain ones while for the out-of-distribution data, we should consider the data representativity. Given our previous findings, the guiding principles of our new metric are twofold: 1) it must consider how much the data distribution has changed (by using an OOD detector) and 2) it should preserve the balance between classes (by comparing the label balance between the selected data and the whole data). Based on these two principles, we propose a distribution-aware test selection metric named *DAT*.

8.5.1 OOD detector

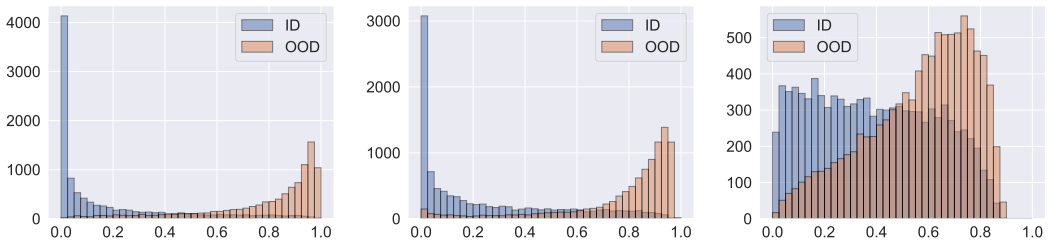
Before looking into DAT, we introduce an OOD detection approach employed in our metric. The outlier exposure (OE) detector [194] is currently the best OOD detection method as assessed in a recent empirical study [47]. Given a distribution $\mathcal{D}_{in}$, the detector aims at identifying if a sample is derived from $\mathcal{D}_{in}$ or not. The main idea is to separately train a DNN which additionally optimizes the loss of OOD data. In real applications, the distribution $\mathcal{D}_{out}$ is unknown and difficult to be inferred precisely. Therefore, in practice, the OOD data (OE dataset, following $\mathcal{D}_{out}^{OE}$) fed into the detector can be the same as or disjoint from the test OOD data. Given a DNN f that learned the distribution $\mathcal{D}_{in}$ and an OE dataset, the objective of the OOD detector is to minimize:

$$\mathbb{E}_{(x,y)\sim\mathcal{D}_{in}}\left[\mathcal{L}\left(f\left(x\right),y\right)+0.5*\mathbb{E}_{x'\sim\mathcal{D}_{out}^{OE}}\left[\mathcal{L}_{OE}\left(f\left(x'\right)\right)\right]\right] \tag{8.3}$$

where $\mathcal{L}$ is the loss function of f. The OE loss function $\mathcal{L}_{OE}$ is set as the cross-entropy from $f(x')$ to the uniform distribution. Particularly, although learning from $\mathcal{D}_{out}^{OE}$, the OOD detector has been proved [194, 47] to generalize well to $\mathcal{D}_{out}$. Our experimental results in Table 8.3 (Section8.3.4) also confirm this conclusion.

Concretely, given a pre-trained model f and its training set $X^{in} \sim \mathcal{D}_{in}$, first, we prepare the ID data and OOD data to train the OOD detector. For image datasets, we use all 8 considered image mutation operators to mutate the training set and generate 8 mutated sets. Then, we evenly select $\frac{|X^{in}|}{8}$ data from each mutated set and combine them as the OOD training set $X^{out} \sim \mathcal{D}_{out}^{OE}$. For the text datasets, we split the data from the OOD set as X^{out} directly. Note that, the OOD data we select for training the OOD detector are not from the candidate set and test data. Next, an OE model is trained using both X^{in} and X^{out} according to Equation 8.3. This model predicts an OE score (probability) of a test being OOD. Finally, we train a regression classifier based on the OE scores of data in X^{in} and X^{out} predicted by the OE model. All the OOD detectors in this paper are available on our project site.[1]

Figure 8.5 shows the distribution of OE scores of three image candidate sets, MNIST, Fashion-MNIST, and CIFAR-10. For the description of candidate sets, please refer to Section 8.3. The blue and orange histograms represent the distributions of the ID and OOD data, respectively. For MNIST and Fashion-MNIST the OOD detector can recognize and separate the ID and OOD data clearly, and the performance on CIFAR-10 is also acceptable. Besides, we calculate the AUC-ROC score of our OOD detectors. The area under the curve of the receiver characteristic operator (AUC-ROC) provides an overall evaluation of the ability of the OOD detector to distinguish between OOD and ID data. A high AUC-ROC indicates good performance. For MNIST, the AUC-ROC scores are 87.74% and 92.69% of LeNet-1 and LeNet-5, respectively. For Fashion-MNIST, the scores are 88.62% and 90.81% of LeNet-1 and LeNet-5, respectively. For CIFAR-10, the scores are 74.52 and 74.36 for ResNet-20 and NiN, respectively.



(a) MNIST, LeNet-5 (b) Fashion-MNIST, LeNet-5 (c) CIFAR-10, ResNet-20

Figure 8.5: Histograms of OE scores of image candidate sets.

8.5.2 DAT Algorithm

In Algorithm 4, we present our proposed DAT selection metric.
Basically, DAT includes five steps to select data:

1. Given a candidate set X_c, we first utilize the OOD detector, $OOD_{Detector}$, to divide this dataset into ID and OOD sets, X_c^{id} and X_c^{out} (line 1). If the OOD score given by the $OOD_{Detector}$ is greater than (less than or equal to) δ, we say the input sample is OOD (ID) data. By default, we set δ to 0.5, however, the

Algorithm 4: DAT: Distribution-Aware Test Selection

Input : $OOD_{Detector}$: out of distribution detector
f, X_t, X_c : DNN, test set, candidate set
$uncertainSelect$: uncertainty-based selection metric
δ: threshold to limit the size of selected ID data
n: size of labeling budget
ite: number of iterations

Output : X_s: selected data

`/* Step1: Check data distribution of` X_c `*/`

$X_c^{in}, X_c^{out} = OE\_Detector(X_c, \delta)$

`/* Step2: Determine data distribution of` X_s `*/`

if $\frac{|X_c^{in}|}{|X_c|} > \delta$ **then**
  $n^{in} = \delta \times n$
else
  $n^{in} = \delta \times \frac{|X_c^{in}|}{|X_c|}$
end
$n^{out} = n - n^{in}$

`/* Step3: Select ID data */`

$X^{in} = uncertainSelect\left(X_c^{in}, n^{in}\right)$

`/* Step4: Select OOD data */`

$Y_t = f\left(X_t\right)$
$LD_t = histogram\left(Y_t\right)$; `// Histogram of labels`
$d_{min} = \infty$
for $i = 0 \rightarrow ite$ **do**
  $X_*^{out} = randomSelect\left(X_c^{out}, n^{out}\right)$
  $LD_r = histogram\left(Y_r\right)$
  if $|LD_t - LD_r| < d_{min}$ **then**
    $X^{out} = X_*^{out}$
    $d_{min} = |LD_t - LD_r|$
  end
end

`/* Step5: Output selected data */`

$X_s = X^{id} \cup X^{out}$
return X^s

appropriate value depends on the used detector (we discuss this in more detail later).

2. From the results of CES, DSA, and random selection in Table 8.7 and Figure 8.4, we know that the selected set, X_s, from X_c should follow a similar data distribution. Thus, we determine the labeling budgets, n^{id} and n^{out}, for the ID and OOD data in X_s by the proportion of ID data in X_c (lines 2-7). Note that, in practice, we select slightly more OOD data like all the selection metrics do because OOD data are more informative for a pre-trained DNN. Here, we use a predefined threshold δ to limit the amount of ID data used for retraining.

3. We first select the ID data. According to our study, we try to select more uncertain data from the ID set. As the result shown in Table 8.7 suggests, DeepGini is appropriate for this because it achieves the highest average top-1 performance among the uncertain-based metrics when the OOD data are below 70%. Thus, we apply DeepGini to select the most uncertain data, X^{id}, from X_c^{id} (line 8).

4. To select the OOD data, we consider the class bias as suggested by RQ3. Using the test data, X_t, as a reference, we select OOD data within several iterations. In detail, first, we create the histogram, LD_t, of the predicted labels, Y_t, of X_t by the pre-trained DNN model f (lines 9-10). In each iteration, a set of OOD data, X_*^{out}, are randomly selected from X_c^{out} (line 13). Based on the distance of histograms between the selected and test sets, X^{out} is updated to be more balanced (lines 11-20).

5. Output the combination of the selected ID and OOD data (lines 21-22).

8.5.3 RQ4: Effectiveness of DAT on Synthetic Distribution Shift

To evaluate the effectiveness of DAT, we conduct a similar comparison as RQ2. First, we consider the synthetic distribution shift. An important component in DAT is the OOD detector which determines the threshold δ to control the size of selected ID and OOD data. Generally, δ is set to 0.5, which means the data sample is OOD (ID) if the detector score of this sample is greater (smaller) than 0.5. However, we experimentally found out that it might be better to set a smaller δ in order to reduce the number of selected ID data. This is because the OOD detector may not be able to perfectly separate ID data from OOD data. In our experiments, we set δ as 0.01, 0.1, and 0.3 for MNIST, Fashion-MNIST, and CIFAR-10, respectively. Besides, we set the iteration number as 1000 for all datasets. For the backbone uncertainty metric that DAT uses to select ID data, we choose DeepGini as discussed before.

Table 8.9 lists the frequency of each selection metric achieving the top-1 and top-3 accuracy improvement over 64 and 192 cases, respectively. On average, DAT is the best metric regardless of the distribution shift and dataset. For example, in the case of "Top-1", DAT is 5 and 2 times better than the worst (Random) and the second-best (MCP), respectively. In the case of "Top-3", although the gap between metrics becomes smaller, DAT still achieves nearly 24% better than the second-best (MCP). Particularly, DAT always outperforms the others when there are more than 70% OOD data. In the other distribution ratios, there is no unique winner while DAT remains generally competitive.

Table 8.9: Effectiveness of DAT: Comparison of frequency of being the top-1 and top-3 best of 7 selection metrics. The best result is highlighted in gray. "**Distribution**" represents different distribution shifts of the candidate set.

	Distribution ID + OOD	Entropy	DeepGini	MCP	CES	DSA	Random	DAT	Entropy	DeepGini	MCP	CES	DSA	Random	DAT
		Top-1							Top-3						
MNIST	0% + 100%	1	0	0	15	3	22	23	2	1	15	48	11	55	60
	10% + 90%	0	0	2	6	1	20	35	2	5	10	46	15	56	58
	20% + 80%	0	2	6	10	3	9	34	2	9	17	39	16	46	63
	30% + 70%	3	4	10	5	0	10	32	9	12	30	39	11	36	55
	40% + 60%	3	4	18	10	5	12	12	16	28	33	33	17	34	31
	50% + 50%	3	6	26	7	2	8	12	20	31	44	25	11	26	35
	60% + 40%	6	8	20	3	5	7	15	28	40	43	18	10	20	33
	70% + 30%	6	12	20	3	4	2	17	50	50	46	7	8	6	25
	80% + 20%	18	12	11	0	1	3	19	55	51	50	3	4	8	21
	90% + 10%	25	16	5	0	2	1	15	56	49	50	0	7	7	23
	100% + 0%	13	7	10	7	3	5	19	30	28	29	26	26	25	28
	Average	7.09	6.45	11.64	6.00	2.64	9.00	21.18	24.55	27.64	33.36	25.82	12.36	29.00	39.27
Fashion-MNIST	0% + 100%	0	1	3	10	8	16	26	1	3	7	45	30	52	54
	10% + 90%	1	0	4	8	8	15	28	1	0	11	40	31	50	59
	20% + 80%	0	0	3	6	8	15	32	1	2	11	35	33	52	58
	30% + 70%	0	2	4	7	8	12	31	4	2	20	38	28	46	54
	40% + 60%	0	0	10	11	6	14	23	3	5	29	40	33	31	51
	50% + 50%	0	2	10	4	5	10	33	5	9	31	40	25	31	51
	60% + 40%	5	7	18	2	1	7	24	14	25	38	20	18	31	46
	70% + 30%	6	12	20	1	1	5	19	26	34	47	17	11	21	36
	80% + 20%	13	15	16	2	0	1	17	40	37	45	12	11	19	28
	90% + 10%	21	14	16	1	1	2	9	49	53	52	6	7	12	13
	100% + 0%	4	7	4	3	2	5	39	31	32	29	18	23	13	46
	Average	4.55	5.45	9.82	7.45	4.27	6.91	25.55	15.91	18.36	29.09	32.09	26.00	25.45	45.09
CIFAR-10	0% + 100%	5	11	4	9	5	13	17	26	29	21	30	19	29	38
	10% + 90%	6	15	3	8	7	6	19	22	34	12	35	19	30	40
	20% + 80%	15	14	0	7	4	4	20	33	35	10	30	18	26	40
	30% + 70%	15	15	1	3	4	10	16	35	37	10	22	15	32	41
	40% + 60%	11	13	4	7	6	7	16	36	40	15	32	20	21	28
	50% + 50%	16	16	1	4	10	4	13	37	44	21	19	19	26	26
	60% + 40%	15	17	4	5	7	4	12	40	38	22	23	21	22	26
	70% + 30%	22	9	6	5	4	4	14	38	45	30	17	15	20	27
	80% + 20%	24	16	5	2	2	5	10	44	47	40	12	12	16	21
	90% + 10%	17	18	5	3	3	4	14	45	48	40	14	10	10	25
	100% + 0%	9	9	18	6	7	4	11	32	32	45	26	17	15	25
	Average	14.09	13.91	4.64	5.36	5.36	5.91	14.73	35.27	39.00	24.18	23.64	16.82	22.45	30.64
Total	0% + 100%	6	12	7	51	34	16	66	29	33	43	136	123	60	152
	10% + 90%	7	15	9	41	22	16	82	25	39	33	136	121	65	157
	20% + 80%	15	16	9	28	23	15	86	36	46	38	124	104	67	161
	30% + 70%	18	21	15	32	15	12	79	48	51	60	114	99	54	150
	40% + 60%	14	17	32	30	23	25	51	55	73	77	95	98	68	110
	50% + 50%	19	24	37	16	16	22	58	62	84	96	92	69	61	112
	60% + 40%	26	32	42	13	9	19	51	82	103	103	62	59	62	105
	70% + 30%	34	33	46	7	9	13	50	114	129	123	43	35	44	88
	80% + 20%	55	43	32	10	2	4	46	139	135	135	36	26	35	70
	90% + 10%	63	48	26	6	4	7	38	150	150	142	23	21	29	61
	100% + 0%	26	23	32	12	15	15	69	93	92	103	58	75	56	99
	Average	25.73	25.82	26.09	22.36	15.64	14.91	61.45	75.73	85.00	86.64	83.55	75.45	54.64	115.00

Besides, to check whether it is important and useful to consider the data distribution in DAT, we conduct an ablation study. Elaborately, we remove the distribution detection steps (steps 1-3) in Algorithm 4 and only use the fourth step to select all the candidate data. In this way, DAT ignores the data distribution. Table 8.10 provides the results of our ablation study. Compared with taking into consideration the data distribution (Table 8.9), the performance drops a lot (presented by the numbers in brackets). On average, the frequencies of being the best top-1 and top-3 have reduced by 12.09 and 11, respectively. This ablation study demonstrates that considering data distribution is critical for DAT.

Answer to RQ4: On the synthetic distribution shift, when there are more OOD data in the new coming set (OOD data $>$ % 70), DAT outperforms other compared metrics in all our considered datasets. In lower ratios of OOD, DAT is not always the best metric but it remains competitive overall. Besides, our ablation study demonstrates the importance of taking into account the data distribution when selecting data.

8.5.4 RQ5: Effectiveness on Natural Distribution Shift

In addition to testing on synthetic distribution shifts, we further evaluate DAT on natural distribution shifts. In our study, we consider 3 datasets, iWildCam, IMDb, and Newsgroups.

Datasets iWildCam is from a recently released benchmark with real-world distribution shifts, WILDs[188]. The shift of iWildCam comes from camera traps. Concretely, researchers collect data using specific camera traps, and then use these data to train an ML model for animal recognition. However, when users deploy this model in the wild, the change of camera traps may cause distribution shifts and harm the performance of the model. In total, iWildCam contains 129809 training data (ID), 14961 OOD validation data, 7314 ID validation data, and 42791 OOD test data. The data are divided into 182 different categories. Please refer to 8.3.2 for details of IMDb and Newsgroups.

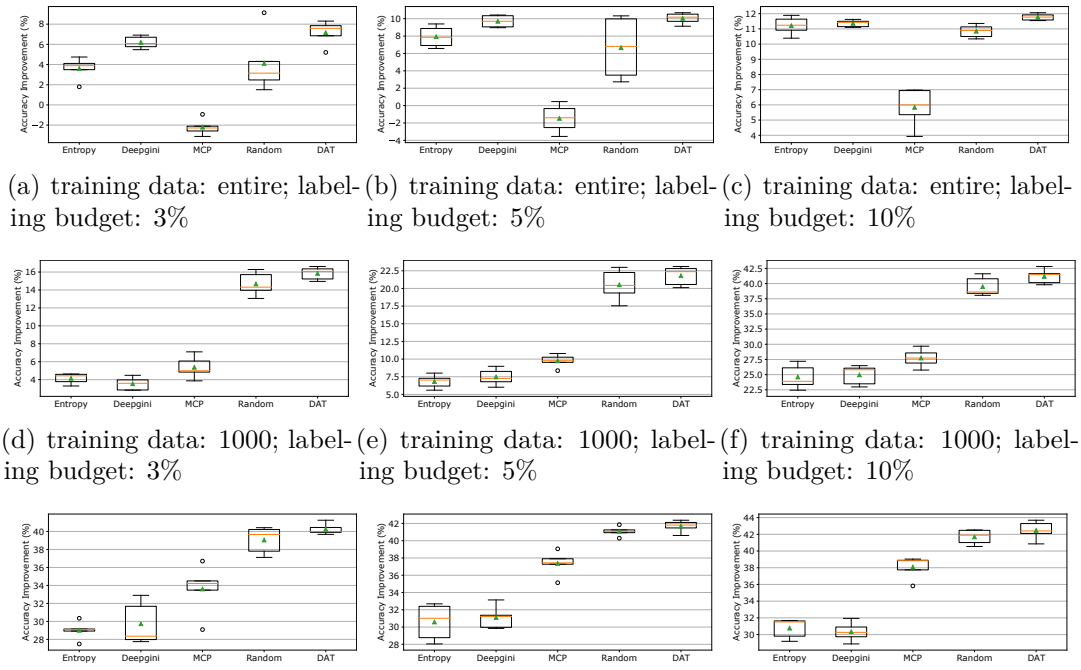
Setup For iWildCam, we use all the training data to train a ResNet-50 model as our pre-trained model. We chose ResNet-50 because it is the recommended model architecture by the WILDs benchmark. Then, we randomly split the test data (all of which are OOD) into three parts, one (20000 data) for training the OOD detector with the ID training data, one (10000 data) as the candidate set for selection, and the rest (12791 data) as the test data. Besides, we follow a similar setup in [197], which reduces the number of training data to check the performance of each metric on the model that has a bad performance, to train models with a small number of training data. In this way, we can check the effectiveness of each metric on both the well-trained model and the model trained by limited labeled data. Thus, we train the other two models using randomly selected 1000 and 2000 training data for our evaluation. For IMDb and Newsgroups, we follow the same procedure as iWildCam to split the OOD data into the training data for the OOD detector, the candidate set for selection, and the test set for evaluation, respectively. After the preparation, we employ different selection metrics to select the candidate data and retrain the pre-trained models. Finally, we record the test accuracy improvement on the test data before and after retraining. This setting is the same as the (0% + 100%) distribution combination in the previous RQs. The AUC-ROC scores of the OOD detectors we trained for this RQ are 79.77%, 68.87%, 70.44%, 82.09%, and

Table 8.10: Ablation study of DAT which shows the importance of considering the data distribution. "**Distribution**" represents different distribution shifts of the candidate set. "**Improvement drop**" presents the drop of accuracy (%) improvement of DAT without the OOD detector compared with using the OOD detector when selecting data. Baseline: Please refer to Table 8.4 for the accuracy of pre-trained DNNs.

	Distribution	**DAT with the OOD detector**		**DAT without the OOD detector**		**Improvement drop**	
	ID + OOD	**Top-1**	**Top-3**	**Top-1**	**Top-3**	**Top-1**	**Top-3**
MNIST	0% + 100%	23	60	19	57	4	3
	10% + 90%	35	58	24	56	11	2
	20% + 80%	34	63	25	57	9	6
	30% + 70%	32	55	25	50	7	5
	40% + 60%	12	31	8	28	4	3
	50% + 50%	12	35	10	33	2	2
	60% + 40%	15	33	14	30	1	3
	70% + 30%	17	25	14	25	3	0
	80% + 20%	19	21	17	20	2	1
	90% + 10%	15	23	14	22	1	1
	100% + 0%	19	28	16	26	3	2
	Average	21.18	39.27	16.91	36.73	4.27	2.55
Fashion-MNIST	0% + 100%	26	54	20	51	6	3
	10% + 90%	28	59	22	55	6	4
	20% + 80%	32	58	30	53	2	5
	30% + 70%	31	54	31	54	0	0
	40% + 60%	23	51	17	49	6	2
	50% + 50%	33	51	25	47	8	4
	60% + 40%	24	46	20	45	4	1
	70% + 30%	19	36	18	30	1	6
	80% + 20%	17	28	13	25	4	3
	90% + 10%	9	13	7	10	2	3
	100% + 0%	39	46	26	38	13	8
	Average	25.55	45.09	20.82	41.55	4.73	3.55
CIFAR-10	0% + 100%	17	38	16	30	1	8
	10% + 90%	19	40	12	33	7	7
	20% + 80%	20	40	16	33	4	7
	30% + 70%	16	41	11	31	5	10
	40% + 60%	16	28	11	24	5	4
	50% + 50%	13	26	9	21	4	5
	60% + 40%	12	26	10	22	2	4
	70% + 30%	14	27	12	24	2	3
	80% + 20%	10	21	10	20	0	1
	90% + 10%	14	25	12	23	2	2
	100% + 0%	11	25	9	22	2	3
	Average	14.73	30.64	11.64	25.73	3.09	4.91
Total	0% + 100%	66	152	55	138	11	14
	10% + 90%	82	157	58	144	24	13
	20% + 80%	86	161	71	143	15	18
	30% + 70%	79	150	67	135	12	15
	40% + 60%	51	110	36	101	15	9
	50% + 50%	58	112	44	101	14	11
	60% + 40%	51	105	44	97	7	8
	70% + 30%	50	88	44	79	6	9
	80% + 20%	46	70	40	65	6	5
	90% + 10%	38	61	33	55	5	6
	100% + 0%	69	99	51	86	18	13
	Average	61.45	115.00	49.36	104.00	12.09	11.00

77.37% for iWildCam-ResNet50, IMDb-LSTM, IMDb-GRU, Newsgroups-NN, and Newsgroups-NN2, respectively. We set the δ in Algorithm 4 for iWildCam, IMDb, and Newsgroups as 0.5, 0.5, and 0.1, respectively.

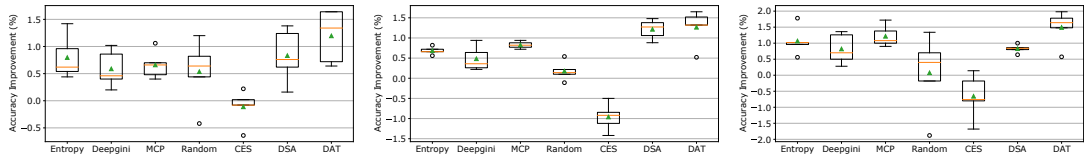
Results Figure 8.6 depicts the accuracy improvement on the test data by using each metric to select (3%, 5%, and 10%) of candidate data for model retraining. Mention that, since both DSA and CES cause out-of-memory problems, we can not run these two metrics on the iWildCam dataset. In the figure, Model-fully, Model-1000, and Model-2000 represent the model that is pre-trained by all training data, 1000 training data, and 2000 training data, respectively. The test accuracy of each model on the test data before retraining is 70.85%, 32.94%, and 35.53% respectively. From the results, we can see that, DAT outperforms the other metrics in all cases. Specifically, on average, DAT can improve test accuracy by 9.25%, 8.60%, 8.65%, and 1.61% more than Entropy, DeepGini, MCP, and Random. When the model is well-trained (using all the training data), in addition to DAT, DeepGini is also a promising metric. However, when the model is trained by limited training data, DeepGini, Entropy, and MCP perform much worse than random selection and DAT. Compared with the random selection, in addition to the higher test accuracy improvement, DAT is also more stable, and the standard deviation of DAT (0.83) is 47% lower than the Random selection (1.56).



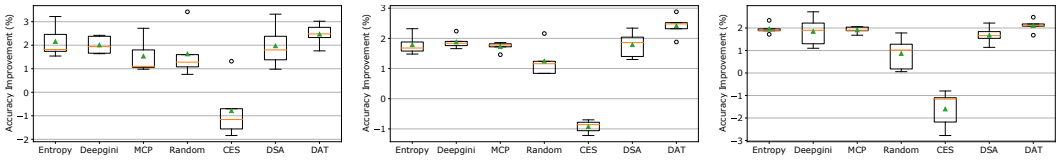
(a) training data: entire; labeling budget: 3%
(b) training data: entire; labeling budget: 5%
(c) training data: entire; labeling budget: 10%
(d) training data: 1000; labeling budget: 3%
(e) training data: 1000; labeling budget: 5%
(f) training data: 1000; labeling budget: 10%
(g) training data: 2000; labeling budget: 3%
(h) training data: 2000; labeling budget: 5%
(i) training data: 2000; labeling budget: 10%

Figure 8.6: Box plot of the accuracy improvement of different selection metrics on the dataset iWildCam, DNN ResNet-50. The pre-trained models are learned by using the entire training set (first row), 1000 data (second row), and 2000 data (third row), respectively. The budgets for retraining are 3% (first column), 5% (second column), and 10% (third column), respectively.

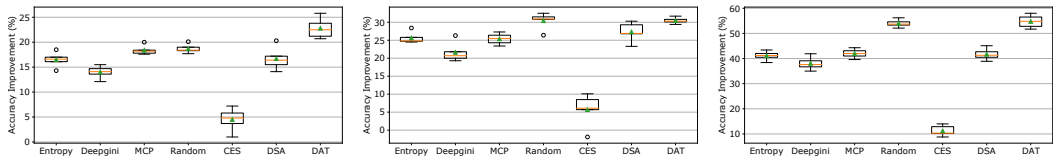
Figure 8.7 shows the results of IMDb and Newsgroups. In most cases (10 out of 12), DAT outperforms the other metrics. On average, for IMDb, DAT can improve



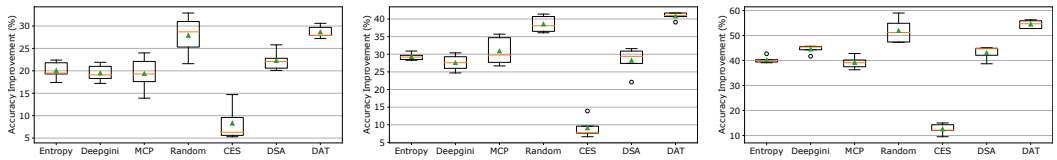
(a) IMDb-LSTM; labeling budget: 3% (b) IMDb-LSTM; labeling budget: 5% (c) IMDb-LSTM; labeling budget: 10%



(d) IMDb-GRU; labeling budget: 3% (e) IMDb-GRU; labeling budget: 5% (f) IMDb-GRU; labeling budget: 10%



(g) Newsgroups-NN; labeling budget: 3% (h) Newsgroups-NN; labeling budget: 5% (i) Newsgroups-NN; labeling budget: 10%



(j) Newsgroups-NN2; labeling budget: 3% (k) Newsgroups-NN2; labeling budget: 5% (l) Newsgroups-NN2; labeling budget: 10%

Figure 8.7: Box plot of the accuracy improvement of different selection metrics on the dataset IMDb and Newsgroups.

the test accuracy by 0.42%, 0.55%, 0.51%, 1.07%, 2.66%, and 0.44% more than Entropy, DeepGini, MCP, Random, CES, and DSA, respectively. The test accuracy improvement seems to be insignificant. We checked the models retrained using all the new data, and the test accuracy improvement is less than 3%. One possible reason is that the natural OOD data for IMDb is similar to the ID data. For Newsgroup, DAT improves the test accuracy by 9.85%, 11.12%, 9.46%, 4.75%, 30.10%, and 8.77% more than Entropy, DeepGini, MCP, Random, CES, and DSA, respectively. Additionally, in this dataset, the Random selection defeats the other metrics except for DAT, which is consistent with our conclusion in RQ2.

Answer to RQ5: In the three datasets with real-world distribution shift, DAT outperforms existing selection metrics by up to 30.10% test accuracy improvement after retraining.

8.6 Discussion

Based on our study, we first highlight our novel findings and research guidance, then discuss the threats to the validity of our work.

8.6.1 Novel Findings and Research Guidance

1. **Retraining process.** Both retraining strategies for model enhancement, 1) only using the selected new data and 2) merging the new data with training data to process retraining, are commonly used in the literature. According to our comprehensive comparison (RQ1), the second process works better. Indeed, only using the new data can improve the accuracy of the new test data, however, the accuracy of the original test set is greatly sacrificed, especially when the new data include more OOD than ID data. By contrast, combining the original training data and new data to retrain a DNN can enhance the performance of the new data, meanwhile, maintaining the high accuracy of the original test set.

   Research Guidance: Based on our experimental results, retraining DNNs by adding new data to the original training set is a better option. There is still room for improving this process. For example, how much original training data is really necessary? Can we reduce the size of the original data to achieve better efficiency? Instead of only selecting new data, proposing a metric to carefully select both the original training data and new data for retraining might be a promising research direction.

2. **Test selection under different data distributions.** Our experiments have demonstrated that none of the existing selection metrics (Entropy, DeepGini, MCP, CES, DSA, and Random) can outperform others under different data distributions. Most of them, Entropy, DeepGini, and MCP, can select useful data for model retraining when the new data contains mostly the ID data. However, for the contrary case where OOD data occupy more of the new data, they fail to win against the random selection. To deal with this specific case, we propose the distribution-aware metric, DAT, and it has been proven to be effective.

   Research Guidance: For model retraining, in the case of more ID data existing in the new data, uncertain-based metrics are better options, while when OOD data are more than ID, our metric DAT can alleviate the influence of distribution shifts and outperform other metrics. Before choosing a metric, the distribution of ID and OOD data should be first checked by some methods, e.g., the OOD detector. However, it is still challenging to develop an almighty metric that can deal with all the data distributions. A promising solution could be considering multiple existing metrics strategically in the retraining process based on the distribution of new data.

3. **Data distribution and bias of selected data.** In terms of data distribution, since OOD data are more likely than ID data to be unlearned by pre-trained DNNs, the uncertain-based selection metrics, Entropy, DeepGini, and MCP, choose more OOD data for retraining under all the different distributions. While CES, DSA, and random selection can follow almost the same data distribution of the candidate data to pick data. On the other hand, concerning the class bias of the selected data, CES and random selection seem to make a good balance among different classes. However, there is no clear clue to show that using more OOD or balanced data is more helpful in model enhancement.

Research Guidance: Concerning the importance of data distribution and class bias, it could be promising to further improve the effectiveness of uncertainty-based metrics (Entropy, DeepGini, and MCP) by considering these two factors. Besides, we observe that most selected data by the uncertainty-based metrics are misclassified by pre-trained DNNs. Thus, the prediction accuracy of the selected data might be another factor that impacts the retraining performance.

8.6.2 Threats to Validity

The external threat to validity mainly comes from the DNN models and datasets used in our study. Regarding the datasets, we consider 6 commonly used and public datasets in the literature. To reduce the threat from the DNN models we employ two well-known architectures for each dataset (except iWildCam, in which we use the state-of-the-art model recommended by WILDs benchmark) to limit the impact of model dependency to some extent. Our datasets include both image and text data, and our models cover both FNN and RNN.

The internal threat could be caused by the implementation of DAT and the selection metrics in comparison. To counter this issue, we borrow the available implementations of the compared methods from released codes by their authors, and carefully check our code.

The construct threat lies in the OOD detector in DAT, the hyperparameter setting, and the randomness in the training process. Following the guidance of a comprehensive empirical study [47], we incorporate the current-best OOD detector into our new metric. Besides, we utilize their implementation and recommend settings to train our OOD detectors. We believe that with a better OOD detector, our method will achieve better results as well. For the hyperparameter, δ is important since it determines how to consider data as ID or OOD. It also limits the ratio of ID data selected for retraining. By default, we set δ to 0.5, that is, up to 50% of the selected data can come from the original distribution. This default ratio worked well for real-world datasets (WILDS and IMDB). For the other datasets, we experimentally found out that setting a lower δ increases the effectiveness of retraining with DAT. Ultimately, at the cost of additional labeling, we can improve the effectiveness of DAT through the setting of a better δ value. In practical applications, this opens the perspective to determine better δ values from past distribution shifts. Regarding the randomness, we repeated each experiment 5 times and we retrained more than 71280 models.

8.7 Conclusion

In this work, we first conducted a systemically empirical study to explore how different retraining processes and data distributions impact the test selection for model enhancement. In total, based on 6 selection metrics in comparison, we retrained 71280 models over 5 popular datasets and 10 DNN models for our empirical study. In terms of enhancing the performance on new data under various distributions and meanwhile maintaining the high accuracy on the original test set, our experimental results reveal that using the combination of training and selected data is better than only using the selected data. Besides, none of the existing selection metrics can always outperform the others across all data distributions. Interestingly, when the new set contains more ($\geq$70%) OOD data, the simple but effective random manner defeats the others, which gives us an insight that this special case has not

been uncovered in existing metrics. Thus, based on the findings, we propose a novel and effective distribution-aware metric, DAT, to deal with this case. In the experiments, we compared DAT with our studied test selection metrics. The results demonstrate that DAT outperforms other metrics by up to five times better for model enhancement when dealing with the synthetic distribution shift. Besides, the results on the datasets with natural distribution shifts also prove that DAT can achieve better model enhancement than the other metrics when facing the in-the-wild scenario. Moreover, based on our findings from the 5 research questions, we open research directions for further improving the performance of existing metrics as well as proposing new selection metrics.

9 Conclusion and Future Work

In this chapter, we conclude this dissertation and present promising future research directions.

Contents

9.1 Conclusion

The dissertation made the first step to label-efficient deep learning engineering. More specifically, it is structured into four parts: 1) label-free model selection; 2) sample-based model training; 3) label-free model evaluation; and 4) label-efficient model enhancement. We now detail them as below.

Regarding the first part, we introduced LAF, a labeling-free technique that targets the task of measuring the performance of multiple DNNs at the same time without labeling effort. LaF bypasses the requirement for domain expertise and simplifies the process of DL engineering. The main idea of this methodology revolves around the construction of a Bayesian model, predicated upon the predicted labels of data, thereby circumventing manual labeling efforts and mitigating the intricacies arising from random sampling. Our evaluation showed that LaF performs well on label-free model evaluation and achieved better results than our considered baselines.

The second part involved a comprehensive empirical exploration of the inherent limitations of active learning. The study reveals that, when employing active learning for model training, the choice of data selection metrics profoundly influences the resultant model quality in terms of both clean accuracy and adversarial robustness. Notably, for image classification tasks, models trained using active learning exhibited competitive test accuracy but suffered from perceptible vulnerabilities in terms of robustness and susceptibility to compression. Furthermore, our contributions extended to pioneering the domain of active code learning, marked by the establishment of a novel benchmark and empirical analysis. Additionally, we proposed to use evaluation metrics as distance calculation methods to enhance active code learning.

For the third part, we proposed Aries to estimate the performance of DNN models on the datasets without label information. We found that models exhibit similar prediction accuracy for data points situated at similar distances from decision boundaries. The finding led Aries to approximate the distances between data points and decision boundaries and learn this information from the existing labeled data to predict the accuracy of new, unlabeled data. Empirical evaluations affirmed the precision and efficacy of Aries, surpassing SOTA baseline methods.

As a final part, we first can in-depth examination of diverse retraining processes and data distribution impacts on sampling-based retraining for model enhancement. Our findings underscored the advantage of combining the training data and selected data to retrain models, outperforming strategies reliant solely on selected data. Notably, when the new set contains $\geq 70\%$ OOD data, random selection defeats all the well-designed methods. Drawing from these insights, we introduced an innovative and more effective metric, DAT, designed to address the challenges posed by data distribution shifts in the context of model enhancement.

9.2 Future Work

We now discuss some promising future directions based on this dissertation.

- **Enhancing LaF and using LaF for other tasks**. A promising direction involves combining existing methods that need to label a part of data and LaF to further improve the effectiveness of model ranking. Recognizing scenarios where LaF might exhibit limitations, such as instances with models of low quality or diversity, introduces the possibility of incorporating a β assessment to gauge model quality. We can use β to check the quality of models first, and

then use sampling-based methods to rank models for these cases. Additionally, the other direction of LaF is to Utilize LaF to build better model ensembles. Since existing works [83] show diversity is a good factor in building ensemble models, it is good to use the β value in LaF to compute the model diversity to help build ensembles.

- **Label-efficient training with multiple objectives.** Our investigation reveals that none of the presently available data selection metrics can perform well on all the considered objectives. This observation opens an intriguing avenue for further research, with a compelling focus on the development of data selection metrics capable of concurrently optimizing multiple multiple model properties such as accuracy, robustness, accuracy after compression, and fairness.
- **Label-efficient testing for LLMs.** Generative models such as large language models (LLMs) have been hot trending recently. How to efficiently estimate the performance of LLMs is an important problem. Consider the scenario of employing ChatGPT for a bug detection task within a substantial project. How can we expediently pinpoint misclassified programs or swiftly gauge ChatGPT's overall project performance to optimize resource allocation? In summary, there are many research and practical opportunities in studying test optimization in DNN in regression and generative tasks. A potential initial step could be adapting existing methods to LLMs and checking their effectiveness.

Bibliography

[1] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[2] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[3] Yaqin Zhou et al. “Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/49265d2447bc3bbfe9e76306ce40a31f-Paper.pdf`.

[4] Qiang Hu et al. “Evaluating the Robustness of Test Selection Methods for Deep Neural Networks”. In: *arXiv preprint arXiv:2308.01314* (2023).

[5] Weijian Deng and Liang Zheng. “Are labels always necessary for classifier accuracy evaluation?” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15069–15078.

[6] Zenan Li et al. “Boosting operational dnn testing efficiency through conditioning”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 499–509.

[7] Junjie Chen et al. “Practical accuracy estimation for efficient deep neural network testing”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29.4 (2020), pp. 1–35.

[8] Qiang Hu et al. “Towards exploring the limitations of active learning: an empirical study”. In: *The 36th IEEE/ACM International Conference on Automated Software Engineering*. 2021.

[9] Qiang Hu et al. “Active Code Learning: Benchmarking Sample-Efficient Training of Code Models”. In: *arXiv preprint arXiv:2306.01250* (2023).

[10] Sara Beery, Elijah Cole, and Arvi Gjoka. “The iWildCam 2020 Competition Dataset”. In: *arXiv preprint arXiv:2004.10340* (2020).

[11] Jianmo Ni, Jiacheng Li, and Julian McAuley. “Justifying recommendations using distantly-labeled reviews and fine-grained aspects”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019.

[12] Pang Wei Koh et al. “WILDS: A Benchmark of in-the-Wild Distribution Shifts”. In: *International Conference on Machine Learning (ICML)*. 2021.

[13] Jie M. Zhang et al. “Machine Learning Testing: Survey, Landscapes and Horizons”. In: *CoRR* abs/1906.10742 (2019). arXiv: 1906.10742. URL: http://arxiv.org/abs/1906.10742.

[14] Lei Ma et al. “Secure deep learning engineering: A software quality assurance perspective”. In: *arXiv preprint arXiv:1810.04538* (2018).

[15] Qiang Hu et al. “Towards Understanding Model Quantization for Reliable Deep Neural Network Deployment”. In: *2023 IEEE/ACM 2nd International Conference on AI Engineering–Software Engineering for AI (CAIN)*. IEEE. 2023, pp. 56–67.

[16] Kexin Pei et al. “DeepXplore: automated whitebox testing of deep learning systems”. In: *Commun. ACM* 62.11 (2019), pp. 137–145. ISSN: 0001-0782. DOI: 10.1145/3361566. URL: https://doi.org/10.1145/3361566.

[17] Lei Ma et al. “Deepgauge: multi-granularity testing criteria for deep learning systems”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018, pp. 120–131.

[18] Yuchi Tian et al. “Deeptest: Automated testing of deep-neural-network-driven autonomous cars”. In: *Proceedings of the 40th International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 303–314.

[19] Zhong Li et al. “Testing DNN-based Autonomous Driving Systems under Critical Environmental Conditions”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 6471–6482.

[20] Augustus Odena et al. “Tensorfuzz: Debugging neural networks with coverage-guided fuzzing”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4901–4911.

[21] Xiaofei Xie et al. “DeepHunter: a coverage-guided fuzz testing framework for deep neural networks”. In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2019, pp. 146–157.

[22] Jianmin Guo et al. “Dlfuzz: Differential fuzzing testing of deep learning systems”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2018, pp. 739–743.

[23] Yang Feng et al. “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*. 2020, pp. 177–188.

[24] Wei Ma et al. “Test selection for deep learning systems”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30.2 (2021), pp. 1–22.

[25] Zan Wang et al. “Prioritizing test inputs for deep neural networks via mutation analysis”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 397–409.

[26] Yu Li et al. “TestRank: Bringing Order into Unlabeled Test Instances for Deep Learning Tasks”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 20874–20886.

[27] Xinyu Gao et al. “Adaptive test selection for deep neural networks”. In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, pp. 73–85.

[28] Junjie Chen et al. “Practical accuracy estimation for efficient deep neural network testing”. In: *ACM Transactions on Software Engineering and Methodology* 29.4 (Oct. 2020). ISSN: 1049-331X. DOI: 10.1145/3394112. URL: https://doi.org/10.1145/3394112.

[29] Yuejun Guo et al. “KAPE: k NN-Based Performance Testing for Deep Code Search”. In: *ACM Transactions on Software Engineering and Methodology* (2023).

[30] Maria E. Ramirez-Loaiza et al. “Active learning: an empirical study of common baselines”. In: *Data Mining and knowledge Discovery* 31.2 (2017), pp. 287–313.

[31] Burr Settles and Mark Craven. “An analysis of active learning strategies for sequence labeling tasks”. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. USA: Association for Computational Linguistics, 2008, pp. 1070–1079.

[32] Zhe Yu, Nicholas A Kraft, and Tim Menzies. “Finding better active learners for faster literature reviews”. In: *Empirical Software Engineering* 23.6 (2018), pp. 3161–3186.

[33] Jinying Chen et al. “An empirical study of the behavior of active learning for word sense disambiguation”. In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. 2006, pp. 120–127.

[34] Fabian Caba Heilbron et al. “What do i annotate next? an empirical study of active learning for action localization”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 199–216.

[35] Manabu Sassano. “An empirical study of active learning with support vector machines forjapanese word segmentation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. 2002, pp. 505–512.

[36] Nadezhda Chirkova and Sergey Troshin. “Empirical study of transformers for source code”. In: *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 2021, pp. 703–715.

[37] Changan Niu et al. “An empirical comparison of pre-trained models of source code”. In: *ICSE 2023* (2023).

[38] Benjamin Steenhoek et al. “An empirical study of deep learning models for vulnerability detection”. In: *ICSE 2023* (2022).

[39] Qiang Hu et al. “CodeS: towards code model generalization under distribution shift”. In: *International Conference on Software Engineering (ICSE): New Ideas and Emerging Results (NIER)*. 2023.

[40] Pengyu Nie et al. “Impact of evaluation methodologies on code summarization”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 4936–4960. DOI: 10.18653/v1/2022.acl-long.339. URL: https://aclanthology.org/2022.acl-long.339.

[41] Haoye Tian et al. “Is ChatGPT the Ultimate Programming Assistant–How far is it?” In: *arXiv preprint arXiv:2304.11938* (2023).

[42] Weisong Sun et al. “Automatic Code Summarization via ChatGPT: How Far Are We?” In: *arXiv preprint arXiv:2305.12865* (2023).

[43] Yiannis Charalambous et al. “A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification”. In: *arXiv preprint arXiv:2305.14752* (2023).

[44] Chunqiu Steven Xia and Lingming Zhang. “Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT”. In: *arXiv preprint arXiv:2304.00385* (2023).

[45] Yinlin Deng et al. “Large language models are edge-case fuzzers: Testing deep learning libraries via fuzzgpt”. In: *arXiv preprint arXiv:2304.02014* (2023).

[46] Wei Ma et al. “The Scope of ChatGPT in Software Engineering: A Thorough Investigation”. In: *arXiv preprint arXiv:2305.12138* (2023).

[47] David Berend et al. “Cats are not fish: deep learning testing calls for out-of-distribution awareness”. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ASE ’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1041–1052. ISBN: 9781450367684. DOI: 10.1145/3324884.3416609. URL: https://doi.org/10.1145/3324884.3416609.

[48] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. “Distribution-aware testing of neural networks using generative models”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 226–237. DOI: 10.1109/ICSE43902.2021.00032.

[49] Niall O’Mahony et al. “Deep learning vs. traditional computer vision”. In: *Advances in Computer Vision*. Ed. by Kohei Arai and Supriya Kapoor. Cham: Springer International Publishing, 2020, pp. 128–144. ISBN: 978-3-030-17795-9. DOI: 10.1007/978-3-030-17795-9_10.

[50] Shervin Minaee et al. “Deep learning–based text classification: a comprehensive review”. In: *ACM Computing Surveys* 54.3 (Apr. 2021). ISSN: 0360-0300. DOI: 10.1145/3439726.

[51] Ruchir Puri et al. *CodeNet: a large-scale AI for code dataset for learning a diversity of coding tasks*. 2021. arXiv: 2105.12655 [cs.SE].

[52] Xinyu Huang et al. “The apolloscape dataset for autonomous driving”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 954–960. DOI: 10.1109/CVPRW.2018.00141.

[53] Uri Alon et al. “code2vec: Learning distributed representations of code”. In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–29. DOI: 10.1145/3290353. URL: https://doi-org.proxy.bnl.lu/10.1145/3290353.

[54] Yan Zheng et al. “Automatic web testing using curiosity-driven reinforcement learning”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 2021, pp. 423–435. DOI: 10.1109/ICSE43902.2021.00048.

[55] Yan Zheng et al. “Wuji: automatic online combat game testing using evolutionary deep reinforcement learning”. In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2019, pp. 772–784. DOI: 10.1109/ASE.2019.00077.

[56] Tianming Zhao et al. “GUIGAN: learning to generate GUI designs using generative adversarial networks”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 748–760. DOI: 10.1109/ICSE43902.2021.00074.

[57] *MLOps*. 2022. URL: https://ml-ops.org/.

[58] *GitHub*. https://github.com/. Online; accessed 25 January 2022. 2022.

[59] Martín Abadi et al. “Tensorflow: a system for large-scale machine learning”. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. OSDI’16. USA: USENIX Association, 2016, pp. 265–283. ISBN: 9781931971331.

[60] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[61] Yan Sun et al. “Improving missing issue-commit link recovery using positive and unlabeled data”. In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2017, pp. 147–152. DOI: 10.1109/ASE.2017.8115627.

[62] *AIZU online judge*. Online; accessed 10 January 2021. 2018. URL: https://judge.u-aizu.ac.jp/onlinejudge/.

[63] David Berend et al. “Cats are not fish: deep learning testing calls for out-of-distribution awareness”. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1041–1052. ISBN: 9781450367684. URL: https://doi.org/10.1145/3324884.3416609.

[64] Rui Hu et al. “Understanding and testing generalization of deep networks on out-of-distribution data”. In: *arXiv preprint arXiv:2111.09190* (2021).

[65] Rob Alexander et al. *Safety assurance objectives for autonomous systems*. Feb. 2020.

[66] Linghan Meng et al. “Measuring discrimination to boost comparative testing for multiple deep learning models”. In: *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2021, pp. 385–396. DOI: 10.1109/ICSE43902.2021.00045.

[67] Omer Sagi and Lior Rokach. “Ensemble learning: a survey”. In: *WIREs Data Mining and Knowledge Discovery* 8.4 (2018). DOI: https://doi.org/10.1002/widm.1249.

[68] Robert L. Ebel. “Procedures for the analysis of classroom tests”. In: *Educational and Psychological Measurement* 14.2 (1954), pp. 352–364. DOI: 10.1177/001316445401400215.

[69] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum likelihood from incomplete data via the em algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38. ISSN: 00359246. URL: http://www.jstor.org/stable/2984875.

[70] Yaqin Zhou et al. "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks". In: *Advances in neural information processing systems* 32 (2019).

[71] Jacob Whitehill et al. "Whose vote should count more: optimal integration of labels from labelers of unknown expertise". In: *Proceedings of the 22nd International Conference on Neural Information Processing Systems*. NIPS'09. Vancouver, British Columbia, Canada: Curran Associates Inc., 2009, pp. 2035–2043. ISBN: 9781615679119. URL: https://proceedings.neurips.cc/paper/2009/file/f899139df5e1059396431415e770c6dd-Paper.pdf.

[72] *Active learning empirical study homepage*. 2021. URL: https://sites.google.com/view/alempirical.

[73] Yann Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324.

[74] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[75] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. University of Toronto, Toronto, 2009.

[76] Norman Mu and Justin Gilmer. "MNIST-C: A Robustness Benchmark for Computer Vision". In: *CoRR* abs/1906.02337 (2019). arXiv: 1906.02337.

[77] Dan Hendrycks and Thomas Dietterich. "Benchmarking neural network robustness to common corruptions and perturbations". In: *Proceedings of the International Conference on Learning Representations* (2019). URL: https://openreview.net/forum?id=HJz6tiCqYm.

[78] *DNN models for Fashion-MNIST*. github.com/Testing-Multiple-DL-Models/SDS/tree/main/models. Online; accessed 2 November 2021. 2021.

[79] Bin Liu. "Consistent Relative Confidence and Label-Free Model Selection for Convolutional Neural Networks". In: *2022 3rd International Conference on Pattern Recognition and Machine Learning (PRML)*. IEEE. 2022, pp. 375–379.

[80] Zenan Li et al. "Boosting operational dnn testing efficiency through conditioning". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, 2019, pp. 499–509. ISBN: 9781450355728. DOI: 10.1145/3338906.3338930.

[81] W. Wayne Daniel. *Applied nonparametric statistics*. Duxbury advanced series in statistics and decision sciences. PWS-KENT Pub., 1990. ISBN: 9780534919764. URL: https://books.google.lu/books?id=0hPvAAAAMAAJ.

[82] *SciPy*. https://scipy.org/. Online; accessed 27 January 2022. 2022.

[83] Yuejun Guo et al. “MUTEN: Boosting Gradient-Based Adversarial Attacks via Mutant-Based Ensembles”. In: *arXiv preprint arXiv:2109.12838* (2021).

[84] Melvin Johnson et al. “Google’s multilingual neural machine translation system: enabling zero-shot translation”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 339–351.

[85] Jens Kober, J Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: a survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[86] Jia Deng et al. “Imagenet: a large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee. 2009, pp. 248–255.

[87] Yuji Roh, Geon Heo, and Steven Euijong Whang. “A survey on data collection for machine learning: a big data-ai integration perspective”. In: *IEEE Transactions on Knowledge and Data Engineering* (2019), pp. 1–1.

[88] Burr Settles. “Active learning”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6.1 (2012), pp. 1–114.

[89] Dan Wang and Yi Shang. “A new active labeling method for deep learning”. In: *International Joint Conference on Neural Networks (IJCNN)*. Beijing, China: IEEE, 2014, pp. 112–119.

[90] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. “Deep bayesian active learning with image data”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1183–1192.

[91] Ozan Sener and Silvio Savarese. “Active learning for convolutional neural networks: A core-set approach”. In: *International Conference on Learning Representations*. Vancouver, BC, Canada: OpenReview.net, 2018.

[92] Melanie Ducoffe and Frederic Precioso. “Adversarial active learning for deep networks: a margin based approach”. In: *arXiv preprint arXiv:1802.09841* (2018).

[93] Xiaofei Xie et al. “DiffChaser: Detecting Disagreements for Deep Neural Networks.” In: *IJCAI*. 2019, pp. 5772–5778.

[94] Xiaoning Du et al. “Deepstellar: Model-based quantitative analysis of stateful deep learning systems”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 477–487.

[95] Jinhan Kim, Robert Feldt, and Shin Yoo. “Guiding deep learning system testing using surprise adequacy”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE. 2019, pp. 1039–1049.

[96] W. Shen et al. “Multiple-boundary clustering and prioritization to promote neural network retraining”. In: *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Melbourne, Australia: IEEE, 2020, pp. 410–422.

[97] Yang Feng et al. “DeepGini: prioritizing massive tests to enhance the robustness of deep neural betworks”. In: *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2020. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 177–188. ISBN: 9781450380089. DOI: 10.1145/3395363.3397357. URL: https://doi.org/10.1145/3395363.3397357.

[98] Wei Ma et al. “Test selection for deep learning systems”. In: *ACM Trans. Softw. Eng. Methodol.* 30.2 (Jan. 2021). ISSN: 1049-331X. DOI: 10.1145/3417330. URL: https://doi.org/10.1145/3417330.

[99] Xiang Gao et al. “Fuzz testing based data augmentation to improve robustness of deep neural networks”. In: *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE. 2020, pp. 1147–1158.

[100] Lin Yang et al. “Suggestive annotation: a deep active learning framework for biomedical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. Cham: Springer International Publishing, 2017, pp. 399–407.

[101] Mengwei Xu et al. “A first look at deep learning apps on smartphones”. In: *The World Wide Web Conference*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 2125–2136.

[102] Tsui-Wei Weng et al. “Evaluating the robustness of neural networks: an extreme value theory approach”. In: *International Conference on Learning Representations*. Vancouver, BC, Canada: OpenReview.net, 2018.

[103] Andrew Maas et al. “Learning word vectors for sentiment analysis”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. 2011, pp. 142–150.

[104] Daniele Vitale, Paolo Ferragina, and Ugo Scaiella. “Classification of short texts by deploying topical annotations”. In: *European Conference on Information Retrieval*. Berlin, Heidelberg: Springer, 2012, pp. 376–387.

[105] Lada A Adamic et al. “Knowledge sharing and yahoo answers: everyone knows something”. In: *Proceedings of the 17th international conference on World Wide Web*. 2008, pp. 665–674.

[106] Shuhuai Ren et al. “Generating natural language adversarial examples through probability weighted word saliency”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 1085–1097.

[107] Yann LeCun et al. “LeNet-5, convolutional neural networks”. In: *URL: http://yann. lecun. com/exdb/lenet* 20.5 (2015), p. 14.

[108] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network”. In: *arXiv preprint arXiv:1312.4400* (2013).

[109] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *International Conference on Learning Representations*. San Diego, CA, USA, 2015.

[110] François Chollet. *Keras code examples*. 2020. URL: https://keras.io/examples/nlp/bidirectional%5C_lstm%5C_imdb/.

[111] Aditya Siddhant and Zachary C. Lipton. "Deep bayesian active learning for natural language processing: results of a large-scale empirical study". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 2904–2909. DOI: 10.18653/v1/D18-1318.

[112] Augustus Odena et al. "TensorFuzz: debugging neural networks with coverage-guided fuzzing". In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 4901–4911.

[113] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "Deepfool: a simple and accurate method to fool deep neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2574–2582.

[114] François Chollet et al. *Keras*. https://keras.io. 2015.

[115] William H Beluch et al. "The power of ensembles for active learning in image classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9368–9377.

[116] Jonas Rauber, Wieland Brendel, and Matthias Bethge. "Foolbox: a python toolbox to benchmark the robustness of machine learning models". In: *arXiv preprint arXiv:1707.04131* (2017).

[117] Maria-Irina Nicolae et al. "ART: adversarial robustness toolbox v1.2.0". In: *CoRR* 1807.01069 (2018). URL: https://arxiv.org/pdf/1807.01069.

[118] Yizhen Dong et al. "There is limited correlation between coverage and robustness for deep neural networks". In: *arXiv preprint arXiv:1911.05904* (2019).

[119] J. Gao et al. "Black-box generation of adversarial text sequences to evade deep learning classifiers". In: *2018 IEEE Security and Privacy Workshops (SPW)*. May 2018, pp. 50–56. DOI: 10.1109/SPW.2018.00016.

[120] Mohit Thakkar. *Beginning machine learning in ios: CoreML framework*. 1st. APress, 2019. ISBN: 1484242963.

[121] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[122] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in Neural Information Processing Systems* 28 (2015), pp. 1135–1143.

[123] Hengyuan Hu et al. "Network trimming: a data-driven neuron pruning approach towards efficient deep architectures". In: *arXiv preprint arXiv:1607.03250* (2016).

[124] Pengfei Li et al. "ACT: an Attentive Convolutional Transformer for Efficient Text Classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 15. 2021, pp. 13261–13269.

[125] Miltiadis Allamanis et al. "A survey of machine learning for big code and naturalness". In: *ACM Computing Surveys (CSUR)* 51.4 (2018), pp. 1–37.

[126] Xing Hu et al. “Summarizing source code with transferred API knowledge”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 2269–2275. DOI: 10.24963/ijcai.2018/314. URL: https://doi.org/10.24963/ijcai.2018/314.

[127] Liuqing Li et al. “CCLearner: a deep learning-based clone detection approach”. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2017, pp. 249–260. DOI: 10.1109/ICSME.2017.46.

[128] Burr Settles. “Active learning literature survey”. In: (2009).

[129] Ozan Sener and Silvio Savarese. “Active learning for convolutional neural networks: a core-set approach”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=H1aIuk-RW.

[130] Michael Weiss and Paolo Tonella. “Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study)”. In: *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)* (2022).

[131] Huan Liu and Lei Yu. “Toward integrating feature selection algorithms for classification and clustering”. In: *IEEE Transactions on knowledge and data engineering* 17.4 (2005), pp. 491–502.

[132] Ruchir Puri et al. “CodeNet: a large-scale AI for code dataset for learning a diversity of coding tasks”. In: *NeurIPS Datasets and Benchmarks*. 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/a5bfc9e07964f8dddeb95fc584cd965d-Abstract-round2.html.

[133] Jeffrey Svajlenko et al. “Towards a big data curated benchmark of inter-project code clones”. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE. 2014, pp. 476–480.

[134] Zhou Yang et al. “Natural attack for pre-trained models of code”. In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, pp. 1482–1493.

[135] Shuai Lu et al. “CodeXGLUE: a machine learning benchmark dataset for code understanding and generation”. In: *CoRR* abs/2102.04664 (2021).

[136] Zhangyin Feng et al. “CodeBERT: a pre-trained model for programming and natural languages”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. DOI: 10.18653/v1/2020.findings-emnlp.139. URL: https://aclanthology.org/2020.findings-emnlp.139.

[137] Daya Guo et al. “Graphcodebert: pre-training code representations with data flow”. In: *arXiv preprint arXiv:2009.08366* (2020).

[138] Yao Wan et al. “Improving automatic source code summarization via deep reinforcement learning”. In: *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. 2018, pp. 397–407.

[139] Donald B Owen. “The power of Student’s t-test”. In: *Journal of the American Statistical Association* 60.309 (1965), pp. 320–333.

[140] Aryaz Eghbali and Michael Pradel. "CrystalBLEU: precisely and efficiently measuring the similarity of code". In: *37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–12.

[141] Shuyan Zhou et al. "CodeBERTScore: evaluating code generation with pretrained models of code". In: (2023). URL: `https://arxiv.org/abs/2302.05527`.

[142] OpenAI. "GPT-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[143] Yu Li et al. "An empirical study on the efficacy of deep active learning for image classification". In: *arXiv preprint arXiv:2212.03088* (2022).

[144] Yuejun Guo et al. "DRE: density-based data selection with entropy for adversarial-robust deep learning models". In: *Neural Computing and Applications* (2022), pp. 1–18.

[145] Yi Sun et al. "Deepid3: face recognition with very deep neural networks". In: *arXiv preprint arXiv:1502.00873* (2015).

[146] Mei Wang and Weihong Deng. "Deep face recognition: a survey". In: *Neurocomputing* 429 (2021), pp. 215–244.

[147] Sorin Grigorescu et al. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386.

[148] Khan Muhammad et al. "Deep learning for safe autonomous driving: current challenges and future directions". In: *IEEE Transactions on Intelligent Transportation Systems* 22.7 (2020), pp. 4316–4336.

[149] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[150] Deheng Ye et al. "Towards playing full moba games with deep reinforcement learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 621–632.

[151] Gregg Rothermel and Mary Jean Harrold. "A safe, efficient regression test selection technique". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 6.2 (1997), pp. 173–210.

[152] Emelie Engström, Per Runeson, and Mats Skoglund. "A systematic review on regression test selection techniques". In: *Information and Software Technology* 52.1 (2010), pp. 14–30.

[153] Qiang Hu et al. "An empirical study on data distribution-aware test selection for deep learning enhancement". In: *ACM Transactions on Software Engineering and Methodology* (2022).

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

[155] Xiyue Zhang et al. "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty". In: *IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE. New York, NY, USA: Association for Computing Machinery, 2020, pp. 739–751.

[156] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. 0. Toronto, Ontario: University of Toronto, 2009.

[157] Kaiming He et al. "Deep residual learning for image recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[158] Ya Le and Xuan Yang. "Tiny imagenet visual recognition challenge". In: *CS 231N* 7.7 (2015), p. 3.

[159] Dan Hendrycks and Thomas Dietterich. "Benchmarking neural network robustness to common corruptions and perturbations". In: *International Conference on Learning Representations* (2019).

[160] Qiang Hu et al. "Deepmutation++: a mutation testing framework for deep learning systems". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2019, pp. 1158–1161.

[161] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. "Deepcrime: mutation testing of deep learning systems based on real faults". In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2021, pp. 67–78.

[162] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[163] Kui Ren et al. "Adversarial attacks and defenses in deep learning". In: *Engineering* 6.3 (2020), pp. 346–360.

[164] Melanie Ducoffe and Frédéric Precioso. "Adversarial active learning for deep networks: a margin based approach". In: *CoRR* abs/1802.09841 (2018). arXiv: 1802.09841.

[165] Jiequan Cui et al. "Learnable boundary guided adversarial training". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 15721–15730.

[166] Javad Zolfaghari Bengar et al. "Reducing label effort: self-supervised meets active learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1631–1639.

[167] Klas Leino, Zifan Wang, and Matt Fredrikson. "Globally-robust neural networks". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 6212–6222.

[168] Jeremy Nixon et al. "Measuring Calibration in Deep Learning." In: *CVPR workshops*. Vol. 2. 7. 2019.

[169] Dan Hendrycks et al. "Augmix: a simple data processing method to improve robustness and uncertainty". In: *arXiv preprint arXiv:1912.02781* (2019).

[170] Dan Hendrycks et al. "PixMix: dreamlike Pictures Comprehensively Improve Safety Measures". In: *arXiv preprint arXiv:2112.05135* (2021).

[171] Daniel Kang et al. "Testing robustness against unforeseen adversaries". In: *arXiv preprint arXiv:1908.08016* (2019).

[172] Yi Sun et al. "DeepID3: face recognition with very deep neural networks." In: *CoRR* abs/1502.00873 (2015). URL: http://dblp.uni-trier.de/db/journals/corr/corr1502.html#SunLWT15.

[173] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[174] Wayne Xiong et al. "Achieving human parity in conversational speech recognition". In: *arXiv preprint arXiv:1610.05256* (2016).

[175] Daisuke Wakabayashi. "Self-driving uber car kills pedestrian in Arizona, where robots roam". In: *The New York Times* 19.03 (2018).

[176] Geoffrey I. Webb et al. "Understanding Concept Drift". In: *CoRR* abs/1704.00362 (2017). arXiv: 1704.00362. URL: http://arxiv.org/abs/1704.00362.

[177] Yang Feng et al. "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*. ISSTA 2020. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 177–188. ISBN: 9781450380089. DOI: 10.1145/3395363.3397357.

[178] Weijun Shen et al. "Multiple-boundary clustering and prioritization to promote neural network retraining". In: *IEEE/ACM International Conference on Automated Software Engineering*. Melbourne, VIC, Australia: Association for Computing Machinery, Dec. 2020, pp. 410–422.

[179] Zenan Li et al. "Boosting operational dnn testing efficiency through conditioning". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, May 2019, pp. 499–509. ISBN: 9781450355728. DOI: 10.1145/3338906.3338930. URL: https://doi-org.proxy.bnl.lu/10.1145/3338906.3338930.

[180] Junjie Chen et al. "Practical accuracy estimation for efficient deep neural network testing". In: *ACM Transactions on Software Engineering and Methodology* 29.4 (Oct. 2020). ISSN: 1049-331X. DOI: 10.1145/3394112. URL: https://doi.org/10.1145/3394112.

[181] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[182] Ken Lang. "Newsweeder: Learning to filter netnews". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 331–339.

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

[184] Xin Li and Yuhong Guo. "Adaptive active learning for image classification". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 859–866. DOI: 10.1109/CVPR.2013.116.

[185] Pál Révész. *Random walk in random and non-random environments*. 2nd. World Scientific, 2005. DOI: 10.1142/5847. eprint: https://www.worldscientific.com/doi/pdf/10.1142/5847. URL: https://www.worldscientific.com/doi/abs/10.1142/5847.

[186] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[187] Rohan Taori et al. "Measuring robustness to natural distribution shifts in image classification". In: *arXiv preprint arXiv:2007.00644* (2020).

[188] Pang Wei Koh et al. "Wilds: A benchmark of in-the-wild distribution shifts". In: *arXiv preprint arXiv:2012.07421* (2020).

[189] Richard Szeliski. *Computer vision: algorithms and applications*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 1848829345.

[190] Kui Ren et al. "Adversarial attacks and defenses in deep learning". In: *Engineering* 6.3 (2020), pp. 346–360. ISSN: 2095-8099. DOI: 10.1016/j.eng.2019.12.012. URL: https://www.sciencedirect.com/science/article/pii/S209580991930503X.

[191] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and harnessing adversarial examples*. 2015. arXiv: 1412.6572 [stat.ML].

[192] Aleksander Madry et al. "Towards deep learning models resistant to adversarial attacks". In: *International Conference on Learning Representations*. Vancouver, BC, Canada, Apr. 2018.

[193] Eric Wong, Leslie Rice, and J Zico Kolter. "Fast is better than free: Revisiting adversarial training". In: *International Conference on Learning Representations*. 2019.

[194] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. "Deep anomaly detection with outlier dxposure". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HyxCxhRcY7.

[195] Bent Fuglede and Flemming Topsoe. "Jensen-Shannon divergence and Hilbert space embedding". In: *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.* IEEE. 2004, p. 31.

[196] Zixun Zhang et al. "MetaSelection: metaheuristic sub-structure selection for neural network pruning using evolutionary algorithm". In: *Frontiers in Artificial Intelligence and Applications* 325 (2020), pp. 2808–2815.

[197] Jannik Kossen et al. "Active testing: Sample-efficient model evaluation". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5753–5763.