








# Federated Learning for Software Engineering: A Case Study of Code Clone Detection and Defect Prediction

Yanming Yang , Xing Hu , *Member, IEEE*, Zhipeng Gao , Jinfu Chen , Chao Ni ,  
Xin Xia , *Member, IEEE*, and David Lo , *Fellow, IEEE*

**Abstract**—In various research domains, artificial intelligence (AI) has gained significant prominence, leading to the development of numerous learning-based models in research laboratories, which are evaluated using benchmark datasets. While the models proposed in previous studies may demonstrate satisfactory performance on benchmark datasets, translating academic findings into practical applications for industry practitioners presents challenges. This can entail either the direct adoption of trained academic models into industrial applications, leading to a performance decrease, or retraining models with industrial data, a task often hindered by insufficient data instances or skewed data distributions. Real-world industrial data is typically significantly more intricate than benchmark datasets, frequently exhibiting data-skewing issues, such as label distribution skews and quantity skews. Furthermore, accessing industrial data, particularly source code, can prove challenging for Software Engineering (SE) researchers due to privacy policies. This limitation hinders SE researchers' ability to gain insights into industry developers' concerns and subsequently enhance their proposed models. To bridge the divide between academic models and industrial applications, we introduce a federated learning (FL)-based framework called ALMITY. Our aim is to simplify the process of implementing research findings into practical use for both SE researchers and industry developers. ALMITY enhances model performance on sensitive skewed data distributions while ensuring data privacy and security. It introduces an innovative aggregation strategy that takes into account three key attributes: data scale, data balance, and minority class learnability.

Manuscript received 6 December 2022; revised 3 October 2023; accepted 13 December 2023. Date of publication 3 January 2024; date of current version 13 February 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62141222, and in part by the National Research Foundation, under its Investigatorship Grant NRF-NRFI08-2022-0002. Recommended for acceptance by A. Zaidman. (*Corresponding author: Xing Hu.*)

Yanming Yang is with the Department of Computer Science and Technology, Zhejiang University, Ningbo 310058, China (e-mail: yanmingyang@zju.edu.cn).

Xing Hu and Chao Ni are with the School of Software Technology, Zhejiang University, Ningbo 315048, China (e-mail: xinghu@zju.edu.cn; chaoni@zju.edu.cn).

Zhipeng Gao is with Shanghai Institute for Advanced Study, Zhejiang University, Hangzhou 310085, China (e-mail: zhipeng.gao@zju.edu.cn).

Jinfu Chen is with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: jinfuchen@whu.edu.cn).

Xin Xia is with the Software Engineering Application Technology Lab, Huawei, Hangzhou 310007, China (e-mail: xin.xia@acm.org).

David Lo is with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: davidlo@smu.edu.sg).

Digital Object Identifier 10.1109/TSE.2023.3347898

This strategy is employed to refine model parameters, thereby enhancing model performance on sensitive skewed datasets. In our evaluation, we employ two well-established SE tasks, i.e., code clone detection and defect prediction, as evaluation tasks. We compare the performance of ALMITY on both machine learning (ML) and deep learning (DL) models against two mainstream training methods, specifically the Centralized Training Method (CTM) and Vanilla Federated Learning (VFL), to validate the effectiveness and generalizability of ALMITY. Our experimental results demonstrate that our framework is not only feasible but also practical in real-world scenarios. ALMITY consistently enhances the performance of learning-based models, outperforming baseline training methods across all types of data distributions.

**Index Terms**—Federated learning, parameter aggregation strategy, skewed data distribution, code clone detection, defect prediction.

## I. INTRODUCTION

ARTIFICIAL Intelligence (AI) has exhibited its formidable prowess in a wide range of research domains, including computer vision [1], speech recognition [2], natural language processing [3], and software engineering [4], [5], [6]. However, learning-based models may encounter challenges when it comes to effective deployment and generalization of their performance in real-world scenarios [7]. This is often due to their conventional training and evaluation solely on specific benchmark datasets, limiting their ability to grasp the nuanced complexities, such as skewed data distributions, commonly encountered in industrial applications [7]. While Software Engineering (SE) researchers have extensively employed various AI techniques to augment developers' productivity, improve software system quality, and enhance decision-making, a notable gap continues to exist between the outcomes of SE academic research and their practical implementation in real-world industrial settings [8]. This disparity manifests in two primary facets:

1) *Due to the data-skewing problem, utilizing academic learning-based models, including machine learning (ML) and deep learning (DL) models, in practice is often ineffective for SE industry practitioners [9].* Real-world industrial data significantly differs from academic benchmarks in terms of volume, variety, velocity, and quality [9]. Academic benchmark datasets are typically derived from general open-source projects (e.g., Apache) and/or platforms (e.g., GitHub), whereas

industrial data is organization-specific, exhibiting distinct data distribution patterns. In various SE fields, industrial datasets often face data-skewing challenges [10]. The two most prevalent data skews for industrial datasets are label distribution skew (imbalanced datasets) and quantity skew (datasets with varying scales) [11]. Specifically, consider the scenario of two companies: one being a large-scale corporation and the other a small-scale enterprise. Both have the shared objective of training a learning-based clone detector capable of autonomously identifying cloned code within their respective software products. However, a substantial scale disparity exists between these two companies, with the large company possessing a code volume approximately 1000 times greater than that of the smaller one. This discrepancy underscores the presence of a quantity skew within their datasets. Furthermore, upon reviewing their code, both companies observe a common pattern: while they acknowledge the existence of a substantial amount of cloned code, these clones only constitute a small fraction, typically ranging about 20%, when compared to the total code volume. This phenomenon is referred to as label distribution skew.

Given the disparity in data distribution between academic and industrial datasets, adapting well-trained academic models to real-world industrial projects becomes challenging. Note that based on the aforementioned findings, our study examines a dataset's data distribution from two perspectives: data scale and data balance degree. For instance, researchers [12] have noted that state-of-the-art academic models exhibit poor performance on industry projects, resulting in a significant performance drop of over 30%. Besides, industry practitioners frequently face constraints in terms of time and cost, making it challenging for them to establish an industrial benchmark dataset on their own. This is especially true for small-scale companies and low-resource organizations that struggle to create sufficiently large datasets for model training. However, traditional training methods prove inadequate in addressing poor model performance arising from dataset skewness and the data hunger problem. While federated learning (FL), a state-of-the-art training method, can aggregate multiple small-scale datasets to train a learning-based model, it is unable to address the issue of low performance caused by skewed datasets. This limitation arises from its aggregation algorithm, which solely considers one attribute: data scale. Therefore, it is imperative to devise a novel training method capable of enhancing model performance on skewed datasets by integrating more informative attributes.

2) *SE academic researchers face formidable challenges, if not outright impossibility, in obtaining access to real-world industrial data due to stringent privacy policies.* Data is commonly recognized as the utmost valuable asset for a software company [13]. Nevertheless, researchers frequently encounter restricted avenues to access industrial data owing to companies' apprehensions regarding data privacy policies. Devoid of exposure to and utilization of real-world industrial data, SE researchers encounter difficulties in effectively applying their well-trained academic models to new datasets, given that the distribution of these datasets may diverge from the benchmarks employed during training. Moreover, this gap is exacerbated by researchers' frequent focus on improving the performance

of state-of-the-art models on benchmark datasets, rather than exploring ways to leverage industrial data for model training while avoiding data leakage. Therefore, devising a method that overcomes data-skewing and data accessibility challenges can prove advantageous and valuable, enabling well-trained academic models to leverage industrial data while mitigating concerns related to stringent data protection policies.

To mitigate the aforementioned challenges, i.e., the impact of skewed datasets on the performance of learning-based models and strict privacy policies, we propose ALMITY, a federated learning (FL)-based framework, aimed at narrowing the gap between SE academic research work and industry applications. This framework enhances the performance of academic models on data with skewed distributions while addressing software practitioners' data security concerns. Specifically, akin to vanilla FL, ALMITY operates with a central server and a few clients, offering a promising solution for safeguarding organizations' sensitive data. It achieves this by sharing solely model updates (e.g., gradient information) and data distribution information instead of raw data. Building upon vanilla FL, ALMITY introduces a novel aggregation strategy that optimizes the model parameters through a comprehensive integration of the model updates based on three distinct attributes: the degree of data **balance**, data **scale**, and **minority** class learnability. By leveraging this strategy, our framework proficiently handles complex and skewed data, effectively mitigating the side-effect of such data on model performance during the model training phase. In essence, our study offers benefits to both SE academic researchers and industry developers seeking to apply or adopt academic research work in industrial practice. More specifically, for industry developers, ALMITY enables the seamless integration of academic research results into their concrete applications. On the other hand, for SE academic researchers, ALMITY enhances the robustness of DL models on industrial datasets without violating strict data protection regulations.

We conduct extensive experiments to verify the effectiveness of ALMITY. Specifically, we focus on multiple significant and extensively studied SE tasks, namely clone detection and defect prediction, as our target tasks. To ascertain the universality of ALMITY, we utilize two distinct types of models, comprising a DL model and a traditional ML model, for evaluating the performance of ALMITY and the respective baselines across each SE task. As ALMITY represents a novel training framework stemming from FL, we employ two mainstream training methods, namely the Centralized Training Method (CTM) and the Vanilla Federated Learning Method (VFL), as baselines for comparison. The experimental results demonstrate the effectiveness of ALMITY in training well-performing ML/DL models on academic benchmarks, affirming its feasibility to tackle SE tasks on academic datasets. To validate the potential capabilities of ALMITY on industrial data, we have not only constructed various datasets with diverse data distributions but also simulated task-specific datasets on their corresponding real-world distributions. In comparison to baseline training methods, ALMITY enables trained ML/DL models to achieve superior performance on datasets with diverse distributions. Specifically, the experimental results clearly demonstrate that ALMITY outperforms the

baselines on all types of skewed data distributions and attains the highest G-Mean performance in both SE tasks. This underscores ALMITY’s potential in enhancing model performance on task-specific datasets while concurrently safeguarding data privacy and security. Taking into account the three attributes employed in our novel aggregation strategy, we have also conducted comprehensive experiments to scrutinize the influence of each attribute on the performance of ALMITY. The experimental outcomes reveal that all three attributes yield a favorable effect on our strategy, with each attribute fulfilling a crucial role in varying data distribution types. In summary, our study makes the following contributions:

- 1) We develop ALMITY, a novel training framework designed to enhance the performance of ML/DL models on sensitive industrial datasets and skewed data distributions while upholding data privacy. To the best of our knowledge, our study is the first work that both employs and enhances FL techniques in SE tasks.
- 2) In contrast to the parameter aggregation method employed in vanilla FL, we introduce two additional essential attributes, i.e., data balance and minority class learnability, to devise a new aggregation method, thereby enhancing ALMITY’s capability to handle skewed data.
- 3) We perform comprehensive experiments on both skewed data distributions and real-world task-specific data distributions to validate ALMITY’s effectiveness in training well-performing models. ALMITY plays a crucial role in bridging the gap between SE academic research and industry applications. Through our study, we aspire to encourage more industry developers to leverage state-of-the-art academic models to enhance their daily development efficiency, while also motivating more academic researchers to create practical models for industrial applications. The replication package of ALMITY is publicly available at: [14].
- 4) We conduct extensive ablation experiments to explore the impact of each attribute utilized in our parameter aggregation strategy on ALMITY’s performance.

The remaining sections of this paper are organized as follows: Section II presents the motivating examples of our study. Section III briefly introduces federated learning. Section IV describes the details of ALMITY. Section V provides the experimental setup, including evaluation tasks and datasets, baseline methods, evaluation metrics, and experimental settings. Section VI illustrates the evaluation process and presents the experimental results. Section VIII and Section IX discuss the lessons learned from this study and the limitations of our approach, respectively. Section X reviews the related work. Finally, Section XI concludes the paper.

## II. MOTIVATING EXAMPLE

In this section, we illustrate the challenges faced by SE researchers and industry developers when applying or adopting ML/DL models in practice. Additionally, we provide motivating examples of using AI for SE in three realistic application scenarios.

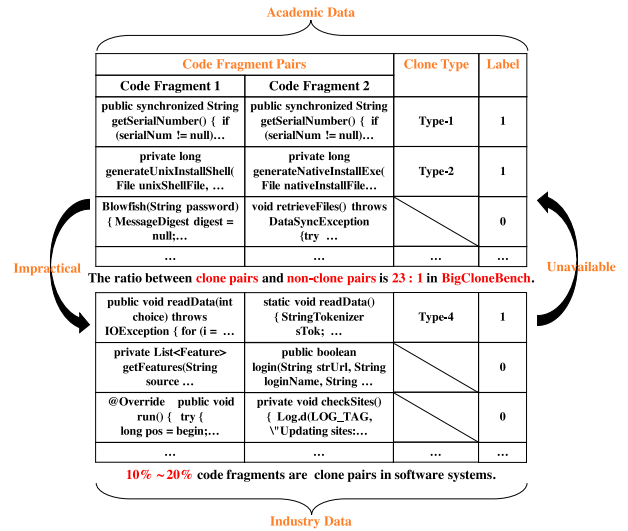


Fig. 1. An illustrating example in code clone detection.

**Scenario one (from industry):** Consider Alice, a SE researcher whose area of interest lies in clone detection, and Bob, a senior industry developer tasked with identifying code clones to maintain code quality within his company. Inspired by the promising performance of DL techniques, Alice devises a novel model for detecting code clones, which, on the widely-used BigCloneBench benchmark dataset, outperformed traditional clone detectors like NiCad [15] and SourcererCC [16]. She is delighted with her research results and proceeds to publish her work in renowned SE conferences or journals. Upon reading Alice’s paper, Bob feels encouraged to adopt this newly released model to enhance code clone detection performance in his daily work. After implementing the model, Bob is surprised to discover that its performance is relatively poor when applied to his industrial data. Through meticulous investigation, he realizes that his real-world industrial data differs drastically from the benchmark dataset in terms of data distribution, including data scale and balance. For instance, the clone ratio in the BigCloneBench dataset is considerably higher compared to the clone ratio in his industrial data. As illustrated in Fig. 1, the benchmark dataset for clone detection shows that the number of real clone pairs is 23 times higher than that of non-clone pairs. In contrast, in industrial datasets, only 10% to 20% of code fragments are likely to be cloned code [17], [18]. Consequently, using a model trained solely on BigCloneBench data without accounting for local data attributes becomes challenging. Furthermore, in addition to attempting to apply Alice’s model directly, Bob also attempts to retrain the model on his local dataset. However, the performance remains unsatisfactory as the model necessitates a large-scale dataset for optimization (BigCloneBench contains over 6 million clone samples), and it is infeasible for Bob to manually label such a vast amount of data samples. Hence, Bob outlines his predicament and seeks assistance and suggestions by emailing Alice.

**Scenario two (from academic):** Upon receiving Bob’s email, Alice acknowledges the limitations of her proposed clone

detection model. However, due to the unavailability of real-world industrial data, she is unable to replicate the experimental results or address the specific issues highlighted by Bob. To develop a more reliable and practical clone detection model suitable for real-world application, Alice recognizes the necessity of exploring real-life data instead of solely relying on the benchmark dataset. However, Bob faces a challenge in granting access to sensitive data due to data privacy and security policies. It becomes exceedingly difficult for Alice to obtain the required access to the sensitive data for her research. Therefore, the challenging task for Alice lies in devising a more potent training method that considers the specific characteristics of industrial data (e.g., data skewness and data hunger problem) when training models, all while ensuring the preservation of sensitive data locally.

**Scenario three (with our framework):** Now, suppose that Bob and Alice adopt our framework, ALMITY. Initially, Alice deploys a shared learning-based model on the central server of our framework. Subsequently, Bob downloads Alice's model and proceeds to train the model locally using their client-side industrial data. Rather than sending sensitive raw data to Alice, ALMITY only packages some insensitive information (including trained model parameters and the number of data instances within each class). Specifically, clients upload their corresponding necessary information to the central server. ALMITY's central server can aggregate the parameters of different clients through the novel parameter aggregation strategy. Later, the server disseminates the aggregated model parameters, which each client then utilizes to update its own model to reach better performance. Finally, Alice provides the model trained by ALMITY to Bob, which improves the clone detection performance on industrial data, enhancing the model's robustness and generalizability in real scenarios.

### III. BACKGROUND

The conventional approach to training learning-based models is centralized, where all datasets are together to train a single model concurrently. This model trained in this method updates its parameters after each training round. Nevertheless, in numerous real-world training scenarios, particularly in industrial applications, datasets from various organizations or companies cannot be amalgamated for training a learning-based model due to the concern about data security and data privacy. This limitation results in suboptimal industrial data utilization and inefficiency.

To address the above issue, Federated learning [19], [20] was introduced as an advanced distributed learning method by Google in 2016, initially intended for transmitting private/sensitive information across various devices [21]. Due to its robust capabilities, federated learning has found wide applications in various areas, including autonomous driving, facial recognition, and system anomaly detection [22], garnering significant attention from both researchers and software practitioners to explore its potential applicability [23], [24].

Federated learning, as a collaboration mechanism, comprises one central server and multiple clients. Each client can be

considered as a company or an organization and independently trains the model on its local dataset. After completing the training rounds, clients will upload their model parameters to the central server in preparation for server-side parameter aggregation. The server adopts a parameter aggregation strategy to generate the optimized parameter by aggregating these client-side model parameters. Subsequently, the server distributes the updated parameter to clients, and thus each client uses the new model parameter to optimize its local model. The performance of the optimized client-side models can exhibit improvements compared to the performance achieved using traditional training methods [25], [26]. Generally, FedAvg [27] is the most common parameter aggregation strategy in federated learning, which solely considers the data scale of client-side datasets to aggregate and update model parameters. The aggregation algorithm of FedAvg is defined as follows:

$$\omega = \sum_{k=1}^K \frac{n_k}{n} \omega_k \quad (1)$$

Here,  $n_k$  represents the count of data instances in client  $k$ , while  $n$  signifies the total count of data instances across all clients.  $\omega_k$  signifies the original model parameter within client  $k$ , and  $\omega$  stands for the updated model parameter resulting from the aggregation of all client-side model parameters.

Upon analyzing Equation 1, it becomes evident that FedAvg confers a heightened degree of decision-making authority and influence to clients equipped with substantial datasets. In simpler terms, the aggregated model parameter gravitates toward the model parameter of the client boasting a large-scale dataset. However, consider a scenario wherein a client possesses an expansive yet notably imbalanced dataset. Unfortunately, this client's model parameter trained on such a heavily skewed dataset fails to contribute effectively to the cultivation of a well-performing model due to its significant deviation from the optimal value. Regrettably, the FedAvg algorithm exacerbates the propensity for the updated model parameter to align itself with the model parameter originating from the client boasting a considerable dataset. This inclination, in turn, steers the updated model parameter away from its optimal value, consequently yielding suboptimal model performance, and even exacerbates the risk of severe model divergence on skewed datasets. Hence, our objective is to rectify this limitation within federated learning, enhancing its suitability for SE tasks through the optimization of the parameter aggregation strategy.

### IV. METHODOLOGY

In this section, we commence by providing an overview of ALMITY, followed by an elaboration of the framework's details.

#### A. Framework Overview

Our proposed paradigm, ALMITY, harnesses the collective potential of multiple client-side datasets to collaboratively train a high-performing model while ensuring data privacy and security. ALMITY comprises three pivotal phases: ① Model Initialization and Training, ② Parameter Uploading and Aggregation,

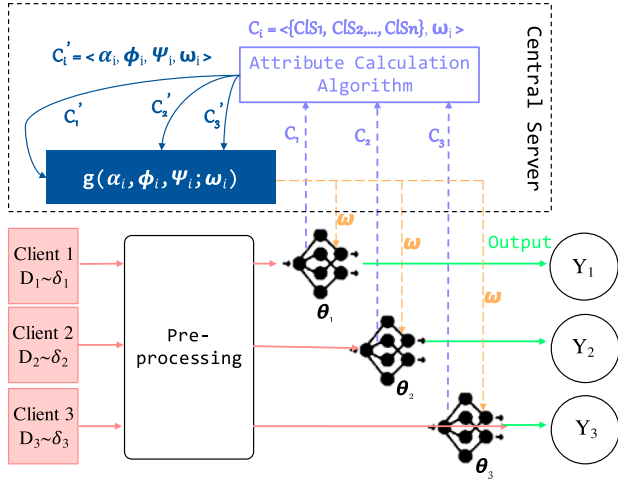


Fig. 2. The workflow of ALMITY.  $\theta$  denotes the shared model in ALMITY.  $C_i$  denotes the “Client-side model info” uploaded to the server from Client  $i$ , including the degree of data balance  $\alpha_i$ , data scale  $\Phi_i$ , and minority class learnability  $\psi_i$ .

and ③ Model Updating and Optimization. To facilitate comprehension, Fig. 2 delineates the workflow of these three phases in intricate detail, represented by the distinct red lines, purple dotted, and yellow dotted lines.

1) *Model Initialization and Training*: The model initialization and training phase encompass two fundamental steps: server-side model initialization and client-side model training.

**Client-side Model Initialization.** For a specific SE task, we commence by meticulously selecting a suitable learning-based model, denoted as  $\theta$ , to serve as the shared model within our framework. This model is deployed on each client, as depicted in Fig. 2. Each client-side model is initiated with the randomly generated model parameter,  $\omega_0$ . Note that the initial model parameters for different client-side models can be different.

**Client-side Model Training.** Each client is in possession of a dataset  $D_i$  with a distinct data distribution  $\delta_i$  (as evident from the red boxes on the client side). Different clients employ a pre-processing technique (as indicated by the red lines on the client side) to convert their respective datasets into a suitable input format for the shared model. In order to maintain decentralization of the training data, client sides execute the initial training round using their respective local datasets after model initialization [28]. Note that the ALMITY framework, characterized as a comprehensive training approach, showcases high scalability in incorporating a wide spectrum of data pre-processing methods for different learning-based models. This integration effectively caters to a variety of data types inherent to different SE tasks.

2) *Parameter Uploading and Aggregation*: At the heart of ALMITY lies this pivotal phase, tasked with producing an enhanced and updated model parameter through the amalgamation of parameters from all individual clients. However, the conventional aggregation strategy employed in the VFL leads to a degradation in model performance when confronted with imbalanced datasets. In order to mitigate the adverse effects of imbalanced datasets on the overarching model performance,

our aggregation strategy introduces three distinct attributes. These attributes serve to characterize both the scale and balance degree of client-side datasets, along with the minority class learnability (i.e., model performance to some extent) of the client-side models. Our proposed aggregation strategy factors in these attributes comprehensively to update the shared model’s parameter. Specifically, following the initial training round, the models trained on the client-side datasets (denoted as  $\theta_1$ ,  $\theta_2$ , and  $\theta_3$  in Fig. 2) exclusively transmit essential and non-sensitive information to the central server. To guarantee data privacy and security without the need to transmit the actual client-side datasets, our framework, ALMITY, only necessitates a minimal amount of information for parameter aggregation. We employ the notation “**Client-side model info**,  $C$ ”,  $C = C_1, C_2, \dots, C_i$ , to symbolize the non-sensitive information transmitted from clients to the server. Thus far, with regard to an individual client, the “Client-side model info”,  $C_i$ , comprises two types of model information: the parameters of the locally trained model ( $\omega_i$ ) and a set of numerical values describing the number of data instances across each class in the corresponding dataset, i.e.,  $C_i = \langle \{Cls_1, \dots, Cls_n\}, \omega_i \rangle$ . Once these two categories of information have been uploaded to the server side, the central server can utilize the attribute calculation algorithms stored on itself to obtain the values of three attributes our parameter aggregation strategy needs to use (see the purple part in Fig. 2): the data scale, data balance, and minority class learnability attributes. Further elaboration on the calculation algorithm for each attribute is presented in Section IV.B.3. Following the attribute calculation step, with respect to a certain client, we change to employ  $C'_i$  to represent the **updated version** of the client-side model information. It encompasses four distinct categories of model information: the parameters of the locally trained model ( $\omega_i$ ), the data scale attribute ( $\Phi_i$ ), the data balance attribute ( $\alpha_i$ ), and minority class learnability attribute ( $\Psi_i$ ); in other words,  $C'_i$  can be expressed as the set  $C'_i = \langle \omega_i, \Phi_i, \alpha_i, \Psi_i \rangle$ . Utilizing the new version of client-side model information ( $C'$ ) as input (see the blue part in Fig. 2), our novel server-side aggregation function ( $g$ ) systematically generates the optimized model parameter (depicted as the yellow  $\omega$  in Fig. 2). In Section IV-B, we delve into the intricacies of our parameter aggregation strategy, elaborating on its specific details and mechanisms.

3) *Client-Side Model Updating and Optimization*: Continuing from the preceding phase, the server proceeds with the dissemination of the updated parameters (depicted as the yellow  $\omega$  in Fig. 2) to the client-side models. Precisely, preceding the next training round, every individual client employs the received model parameter to update its respective local model. At this juncture, all client-side models share an identical model parameter, i.e., the aggregated model parameter. Subsequently, these clients embark on the next training round, resulting in the acquisition of fresh respective client-side model parameters. At the commencement of a new training round, the above three phases are sequentially executed once again. With the progression of iterations, the updated model parameter steadily converges toward an optimal solution. This iterative refinement serves to alleviate the detrimental effects stemming from

skewed datasets and contributes to elevating the performance of client-side neural models to higher levels of proficiency. Aided by ALMITY, every client is empowered to convert a shared learning model into a well-trained local model through the utilization of its own localized data, all the while upholding the privacy of the original sensitive data.

### B. Parameter Aggregation Strategy

The paramount component of our paradigm, the parameter aggregation strategy, centers on the computation of the optimal aggregated parameter for the shared model. This is achieved through the astute fusion of parameters derived from distinct client-side models, aided by the utilization of three distinct attributes. In the next subsection, we provide an exhaustive introduction to our aggregation algorithm.

1) *Problem Formulation:* For FL techniques, the overarching framework for formulating parameter aggregation strategies is generally established as follows:

$$\omega = \sum_{k=1}^K \frac{\beta_k}{\beta} \omega_k \quad (2)$$

Where  $\omega_k$  represents the model parameter of Client  $k$ , while  $\omega$  signifies the optimized parameter of the shared model, achieved through the aggregation of all individual client-side model parameters. The symbol  $\beta_k$  denotes the weight allocated by an aggregation algorithm to the model parameter of Client  $k$ . We can find from this equation that a client-side model parameter assigned a substantial weight implies its proximity to the optimal value, and this parameter becomes a prominent reference for enhancing the overall model performance. In addition, different aggregation strategies allocate varying weights (i.e.,  $\beta_k$ ) to individual client models depending on the factors they take into account. As an illustration, Equation 1 illustrates the parameter aggregation algorithm employed in the standard FL framework. This aggregation strategy solely takes into consideration the scale of local data on the client side when assigning weights to respective client-side models. This indicates that the aggregated parameter resulting from this strategy closely approximates the model parameter of the client with an extensive dataset. However, when large-scale datasets within such clients exhibit significant imbalance, the effectiveness of this strategy often diminishes in yielding the optimal model parameter. This, in turn, leads to poor model performance. To address this issue, we formulate an innovative aggregation strategy that considers two novel factors: data balance and minority class learnability, in addition to data scale. In the following section, we provide a comprehensive description of the new aggregation strategy.

2) *Strategy Description:* Our aggregation strategy synergistically integrates three attributes: data scale, degree of data balance, and minority class learnability. These attributes collectively evaluate the significance of the model parameter within each client to the overall model performance, thereby facilitating the generation of aggregated model parameters. We hold the belief that the combined influence of these three attributes significantly impacts the process of aggregating client-side model parameters.

**The motivation behind introducing the data balance attribute.** Given the inclusion of an attribute in the initial aggregation strategy, which aims to depict the scale of client-side datasets (i.e., the quantity skew), our intuition dictates the necessity of introducing an additional metric. This metric is meant to measure the label distribution skew (referred to as the data balance degree) within these datasets, acting as a complementary factor to assess the quality of client-side datasets. We believe that on the condition of two datasets with the same data scale, a more imbalanced one should be assigned a smaller weight during the parameter aggregation. This is because the more imbalanced dataset plays a negative role in client-side model performance, making its model parameters low reference value. Therefore, our aggregation algorithm integrates the data balance degree as a new attribute that can impact the optimal model parameter generation.

**The motivation behind introducing the minority class learnability attribute.** Besides, accounting for data scale and data balance attributes, the learnability of the minority class within a dataset should also be regarded as a third factor when updating the shared model parameters. This insight arises from the fact that the data scale and data balance alone cannot exclusively dictate the performance of these models although these two attributes are crucial for training well-performed models. Another pivotal factor influencing the performance of learning-based models is the availability of a sufficient number of data samples within datasets, particularly for the minority class, in order to effectively support the accuracy of the model's learning process. As an illustration, considering the well-known large-scale code clone benchmark called BigCloneBench, this dataset comprises a significantly higher count of cloned code pairs (6,164,953) compared to non-cloned pairs (258,574), which is nearly 24 times larger than the number of non-cloned pairs. In general, learning-based models trained on imbalanced datasets struggle to acquire knowledge from the minority class, frequently resulting in the classification of all test samples as belonging to the majority class. However, despite the apparent imbalance in this dataset, many learning-based clone detectors are able to achieve exceptional performance on it. This success can be attributed to the inclusion of a substantial number of minority class samples within BigCloneBench, thereby facilitating models in getting the capacity to differentiate between clone and non-clone code pairs. In other words, these datasets are recognized for maintaining a substantial level of learnability within their minority classes, enabling a multitude of models trained on such imbalanced data to reach noteworthy levels of performance. Therefore, expanding upon this observation, we introduce an additional attribute referred to as "minority class learnability". This attribute is carefully crafted to evaluate the influence of datasets on model performance by quantifying the degree to which the minority class can be effectively learned within the client-side dataset.

To validate the accuracy of our perspectives, we undertake a user study to confirm the correlation and importance among these three attributes in relation to the parameter aggregation process. Specifically, we extend invitations to 11 researchers who possess substantial expertise in utilizing learning-based

models and have applied FL-based techniques. Initially, we outline the workflow of the parameter aggregation strategy employed in the conventional FL technique. Subsequently, we highlight the limitations of this approach and introduce the attributes (i.e., data scale, data balance, and minority class learnability) that, in our view, warrant consideration when executing parameter aggregation. In our user study, these invited 11 well-experienced researchers need to independently evaluate the significance of these three attributes in relation to the parameter aggregation process, leveraging their professional expertise. They are requested to assign a score, ranging from 1 to 5, to each of the attributes for evaluation purposes. A rating of 1 indicates the lowest importance for the evaluated attribute and a rating of 3 indicates some degree of importance for this attribute, whereas a rating of 5 indicates the substantial significance of this attribute to parameter aggregation. Upon analyzing the participants' outcomes, we ascertain that all 11 researchers have assigned scores of no less than 3 to these three attributes—namely, data scale, data balance, and minority class learnability. This indicates their collective belief in the significance of these factors within the parameter aggregation process.

Given the paramount significance of these three attributes in the parameter aggregation process, we incorporate each of them equally into our aggregation strategy. Therefore, the formulation of our parameter aggregation strategy is as outlined below by integrating these three attributes:

$$\beta_k = \alpha_k \times \Phi_k \times \Psi_k \quad (3)$$

Within this equation,  $\beta_k$  signifies the weight allocated to the model parameter in Client  $k$ , which is determined by our aggregation strategy. Meanwhile,  $\alpha_k$  reflects the extent of data balance in the dataset of Client  $k$ .  $\Phi_k$  symbolizes the scale of the dataset in Client  $k$ , while  $\Psi_k$  encapsulates the minority class learnability of Client  $k$ 's dataset.

3) *Attributes Calculation Algorithms*: In this section, we provide a comprehensive introduction of the three aforementioned attributes utilized in our aggregation strategy, including the data scale ( $\Phi$ ), data balance ( $\alpha$ ), and minority class learnability ( $\Psi$ ).

**Data Scale Attribute ( $\Phi$ )**. Large-scale datasets are advantageous for training high-performing learning-based models. Following this conventional wisdom, model parameters derived from extensive client-side datasets tend to approach the optimal parameters more closely than those from other clients. Hence, clients with larger dataset scales should wield a more substantial impact on the generation of updated model parameters. To precisely compute a dataset's scale and incorporate this attribute into our aggregation algorithm, we employ the following equation to assess the data scale attribute  $\Phi$ :

$$\Phi_k = \frac{n_k}{n} \quad (4)$$

Here,  $n_k$  signifies the number of data instances in Client  $k$ , while  $n = \sum_{k=1}^K n_k$  corresponds to the overall data scale across all clients. In the formula 4,  $\Phi_k$  signifies the dataset's scale proportion in Client  $k$  in relation to the collective data volume across all clients.

**Data Balance Attribute ( $\alpha$ )**. Our aggregation strategy uses the attribute  $\alpha$  to describe the degree of data balance for each client-side dataset. To make our strategy more universally applicable and persuasive, we employ Shannon entropy [29] to compute the balance level of datasets, a method widely recognized for measuring data balance [30]. Specifically, the balance of a dataset can be quantified using the following measure:

$$\alpha_k = \frac{H}{\log k} = \frac{-\sum_{i=1}^k \frac{c_i}{n} \log \frac{c_i}{n}}{\log k} \quad (5)$$

Assuming a dataset comprises  $n$  instances with  $k$  classes, where each class has a size denoted as  $c_i$ , the degree of balance within this dataset can be calculated by  $\alpha_k$ . The value of  $\alpha_k$  falls within the range of 0 to 1, with 0 signifying an extremely unbalanced dataset and 1 indicating a well-balanced one. The greater the value of  $\alpha$ , the more balanced the dataset becomes.

**Minority Class Learnability Attribute ( $\Psi$ )**. To bolster the ability of learning-based models to distinguish between different classes, it is advantageous for a dataset to contain as many minority class samples as feasible. This is because a dataset rich in minority-class instances is more likely to assist models in characterizing instances across various classes. Therefore, such datasets with more minority class instances should be assigned substantial weights in the parameter aggregation process. To accomplish this, we introduce a novel formula for assessing the learnability of the minority class within a dataset. More precisely, building upon our insight, we transform the measurement of minority class learnability of each client into calculating the proportion of minority class instances within each dataset accounting for the total number of this class instances across all client-side datasets. Note that, given that ALMITY is a unified learning framework encompassing multiple client sides, the minority class is defined as the class with the fewest instances among all client-side datasets. In particular, the attribute of minority class learnability ( $\Psi$ ) is quantified using the following formula:

$$\Psi_k = \frac{mc_k}{mc} \quad (6)$$

Here,  $mc$  denotes the total count of minority class instances across all client-side datasets;  $mc_k$  signifies the number of minority class instances within the dataset of Client  $k$ .

## V. EXPERIMENT SETUP

In this section, we present the experimental setup, encompassing evaluation tasks, model selection, datasets, baselines, and experimental settings.

### A. Evaluation Downstream Tasks

We have selected two prominent SE classification tasks, i.e., defect prediction (DP) and code clone detection (CCD), as our case studies for evaluating our approach. These choices are motivated by several factors: 1) DP and CCD represent two of the most well-established and extensively researched SE tasks by previous studies [16], [31], [32], [33], [34]. 2) Both DP and CCD tasks provide readily available benchmark

datasets, such as BigCloneBench and the dataset from Devign [35]. These datasets are widely recognized and employed by academic SE researchers. 3) There exists a substantial divergence in data distribution between real-world data and the commonly used benchmarks in DP and CCD [17], [36]. This divergence emphasizes the significance of evaluating our approach across these tasks, as it highlights the challenges posed by real-world data.

**Defect prediction Task.** This task aims to predict vulnerable code fragments from software projects, which can be formulated as a binary classification problem. Let a dataset can be defined as  $\langle (c_i, y_i) | c_i \in C, y_i \in Y \rangle (i \in 1, 2, \dots, n)$ , where  $C$  represents code fragments or metrics,  $Y = \{0, 1\}$  denotes the label set, and  $n$  is the number of samples. Similarly, for each code fragment  $c_i$ ,  $y_i$  represents whether  $c_i$  is vulnerable (label 1) or not (label 0).

**Code clone detection task.** This task aims to detect code fragments that are similar to each other or implement the same functionality. In clone detection studies, a code pair is commonly represented by two vectors  $v1$  and  $v2$ . The distance between the two code fragments can be measured by estimating their distance  $d = |v1 - v2|$ . Thus we can assign the output  $\mathbf{y} = \text{sigmoid}(\mathbf{x}) \in [0, 1]$  according to their similarity, where  $\mathbf{x} = W_o d + b_o$ . The output  $\mathbf{y}$  represents the probability of whether two code fragments are clones (label 1) or not (label 0).

## B. Model Selection

To assess the generalizability and effectiveness of ALMITY across various types of learning-based models, we select diverse models, comprising one DL-based and one ML-based model, to evaluate the performance of ALMITY in these two evaluation tasks.

**DL-based model.** Recently, the field of deep learning has seen significant advancements with the emergence of large-scale pre-trained models such as BERT [37]. SE researchers have also explored the effectiveness of applying these pre-trained models to various SE tasks, including clone detection [38] and defect prediction [39]. Drawing inspiration from the remarkable success of BERT, Microsoft researchers have developed a large-scale pre-trained model called CodeBERT [37]. CodeBERT is designed to handle both natural language and programming language tasks, making it a versatile choice as the foundation for various downstream SE tasks, including clone detection and defect prediction. In this study, we utilize CodeBERT as the basis for our code clone detector [40] and defect predictor [35] for the two aforementioned evaluation tasks.

**ML-based model.** As a versatile training framework, ALMITY can be effectively applied to ML-based models as well. To demonstrate this, we employ one of the most widely utilized ML techniques, Support Vector Machine (SVM), in our study and utilize SVM for defect prediction and clone detection tasks, respectively. Our selection of SVM is driven by two reasons: 1) In previous studies [41], [42], SVM, as a classification algorithm, has been utilized in both of these evaluation tasks. 2) Among various ML techniques,

SVM stands out as one of the algorithms compatible with an FL-based framework.

## C. Datasets

During the evaluation phase, our aim is to employ ALMITY for training DL-based and ML-based models for two SE tasks, thereby verifying the effectiveness of ALMITY. In pursuit of this objective and to showcase the generalizability of ALMITY, we select two distinct datasets for the defect prediction and clone detection tasks, respectively. In each SE task, one dataset is allocated for training the DL-based model, while the other dataset is designated for training the ML-based model. The datasets used in our study originate from diverse sources, encompassing different real-world projects, code sourced from international programming competitions, etc.

### 1) Defect Prediction:

**Dataset used in training the DL model.** CodeXGLUE, as one of the most well-known benchmarks, includes a commonly used dataset for defect prediction. The dataset has been applied in many previous studies [35] and contains 27,318 function-level code fragments. 12,460 code fragments are vulnerable and the remaining code fragments are not.

**Dataset used in training the ML model.** PROMISE can be recognized as one of the most frequently employed defect prediction datasets. This dataset we select includes defect data from 11 distinct Java projects in 41 different versions [43].

### 2) Clone Detection:

**Dataset used in training the DL model.** We employ the widely-used clone dataset known as BigCloneBench (BCB) [44] for clone detection. BigCloneBench encompasses established clones sourced from the IJaDataset repository, with each code fragment pair meticulously categorized into their respective clone types. BigCloneBench comprises a vast collection of over 6,000,000 function-level clone pairs extracted from 25,000 systems. This dataset includes 16,168 Type-I clones, 3,733 Type-II clones, 11,286 strongly Type-III clones, 53,880 moderately Type-III clones, along with 6,079,886 instances of weakly Type-III & Type-IV clones at the method level. Therefore, in order to assess the effectiveness of ALMITY on a smaller-scale dataset, we adopt the approach presented in [40], which involves scaling down the BCB benchmark to mimic real-world dataset conditions. In our commitment to maintaining the integrity and accuracy of the evaluation, we adhere to the dataset distribution employed in the prior study [40], reducing the dataset's scale until it closely resembles the defect prediction dataset in terms of data scale. Finally, our study employs the dataset that aligns with the dataset in CodeXGULE [40], comprising a total of 25,027 code pairs, of which 12,523 pairs are identified as clones.

**Dataset used in training the ML model.** To assess the generalizability and effectiveness of ALMITY across diverse datasets, we introduce another dataset - Google Code Jam dataset [45]. This dataset is a also well-established choice in the field of code clone detection and serves as an essential benchmark for our assessment [5], [46]. Google Code Jam comprises programs gathered from an online programming competition



hosted by Google. For the same rationale, we continue to adopt the approach outlined in [40] to reduce the dataset's scale, resulting in 29, 542 code pairs with 13, 542 clone pairs.

#### D. Baselines

Given that ALMITY represents an innovative Federated Learning (FL)-based training framework, we consider the following two training methods as baselines in our study:

- i) *Centralized Training Method (CTM)*. It refers to the traditional training method, i.e., training the model locally.
- ii) *Vanilla Federated Learning Framework (VFL)* [47]. It refers to the typical federated learning framework with Fedavg aggregation strategy [27].

Moreover, recognizing that numerous techniques, such as oversampling, can effectively mitigate skewness by adjusting dataset instances and bolstering local models, we integrate the oversampling technique into all different training methods. Consequently, in order to maintain as much consistency as possible in input data across different training methods, we proceed to evaluate the performance of ALMITY, VFL, and CTM when incorporating the oversampling technique respectively, i.e., *ALMITY + oversampling*, *VFL + oversampling*, and *CTM + oversampling*. Note that the reason for not adopting the undersampling technique is that it reduces a portion of data instances from the original training dataset to achieve balance, which may result in the loss of valuable data instances.

#### E. Experimental Settings

CTM, FL, and ALMITY are all trained on a Linux server equipped with two NVIDIA GeForce RTX 3090Ti GPUs, each boasting 24 GB of memory. Throughout the training process, we adhere to the parameter configurations detailed in previous research [40], including a batch size and learning rate set at  $2^{-5}$ , etc. To ensure a fair and equitable comparison, all three training methods undergo an identical number of training rounds. In our study, we allocate 80% of the data instances within each client-side dataset for training, while reserving 10% for validation and 10% for testing, in accordance with established practices [40]. For a comprehensive breakdown of the parameter settings, please refer to our replication package.

## VI. EXPERIMENTAL RESULTS

To better understand ALMITY's effectiveness in training well-performed models, we analyze our evaluation results by addressing four research questions (RQs):

- 1) RQ-1: How effective is ALMITY on academic datasets across various SE tasks?
- 2) RQ-2: Can ALMITY outperform baselines on diverse skewed data distributions?
- 3) RQ-3: How do individual attributes affect ALMITY's effectiveness?

#### A. RQ-1: How Effective Is ALMITY on Academic Datasets Across Various SE Tasks?

**Motivation.** To the best of our knowledge, federated learning techniques have seldom been applied to conventional SE

tasks in previous research. Hence, this RQ seeks to accomplish two main objectives: 1) Assess the viability and practicality of employing FL-based approaches, including FL and ALMITY, for addressing traditional SE tasks. 2) Examine and compare the performance of various training methods when applied to academic datasets. We conduct experiments in two extensively studied SE tasks, i.e., code clone detection and defect prediction, to validate the effectiveness and efficiency of ALMITY.

**Method.** In contrast to CTM, both FL and ALMITY represent distributed training methods, featuring a single server and multiple clients. In this RQ, each client serves as a simulated participant, holding a segment of the academic dataset. All participants collaborate by contributing their local datasets to collectively train a learning-based model while ensuring data privacy is upheld. In our experiments, the number of clients ( $N$ ), varies from two to six to evaluate the performance of different training methods due to computational resource constraints. Using four clients as an illustrative case, our experimental procedure adheres to the original data balance criteria, partitioning the complete academic dataset into four equally sized client-side datasets. Subsequently, each client-side dataset is further segmented into training, validation, and testing subsets according to an 8:1:1 ratio. It is important to note that the scale of the client-side datasets utilized for local model training fluctuates depending on the number of clients involved. For instance, when  $N = 4$ , each client employs one-quarter of the entire dataset to construct its local model. Furthermore, in order to ensure an equitable performance comparison between the CTM and FL-based methods, we provide not only the model's average performance trained on client-side datasets using the CTM method (i.e.,  $Client_{client}$  in Table I and Table II) but also the performance achieved by the model trained on the complete dataset (i.e.,  $Client_{all}$  in Table I and Table II).

Furthermore, to comprehensively assess the effectiveness of ALMITY, we extend our evaluation beyond DL models to encompass ML models. We conduct comparative experiments to gauge how different training methods impact the performance of ML models. Since many ML models rely on feature engineering, we find it necessary to extract a set of features from ML-based datasets, specifically for defect prediction and clone detection tasks. Handling the defect dataset is relatively straightforward, as it is already metric-based, encompassing 22 distinct defect metrics. However, unlike defect datasets, metric-based clone detection datasets are scarce. This scarcity arises because clone detectors primarily assess whether a pair of code fragments are cloned by comparing their source code, resulting in datasets primarily composed of code fragments rather than code metrics. While a few prior studies [48] have proposed metrics for clone detection, these metrics often struggle to fully capture both code semantics and structure and are not uniform in different studies. In light of this, our study opts to consider the representation of a code fragment generated by GraphCodeBERT [38], which excels in encapsulating both code semantics and structure, as the code metric used for training an ML-based clone detector. This representation allows us to delve into the evaluation of ALMITY on the clone detection task more effectively.

TABLE I  
THE PERFORMANCE OF DIFFERENT TRAINING METHODS IN THE CONTEXT OF CODE CLONE DETECTION (CCD) TASK

Model	#Client	2							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+Oversampling	ALMITY+Oversampling	CTM <sub>client</sub> +Oversampling	CTM <sub>all</sub> +Oversampling
DL	Precision (%)	97.7	97.1	77.7	88.0	96.6	98.3	79.2	90.5
	Recall (%)	90.8	92.6	77.1	93.9	93.1	94.0	85.0	95.8
	F1-score (%)	94.1	94.8	77.4	90.1	96.1	94.8	82.3	92.7
	AUC (%)	94.5	95.1	78.5	91.2	95.2	96.3	83.2	93.6
	G-Mean (%)	94.4	95.0	78.5	91.2	95.1	96.3	83.2	93.5
	Pfa (%)	2.4	1.9	20.0	21.4	2.9	1.4	21.0	2.5
ML	Precision (%)	56.5	58.5	60.6	45.2	53.0	54.6	59.7	58.3
	Recall (%)	32.8	46.6	24.2	38.0	36.1	61.6	24.6	35.0
	F1-score (%)	41.5	51.9	34.5	41.0	43.0	57.9	34.9	49.9
	AUC (%)	55.7	59.3	55.4	53.3	54.5	59.1	55.3	56.7
	G-Mean (%)	50.8	53.9	45.8	51.2	51.3	54.1	46.0	51.4
	Pfa (%)	21.4	28.0	13.3	26.3	27.0	33.3	14.1	21.5
Model	#Client	3							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+OS	ALMITY+OS	CTM <sub>client</sub> +OS	CTM <sub>all</sub> +OS
DL	Precision (%)	93.8	97.8	70.1	90.0	95.0	96.5	66.7	90.5
	Recall (%)	91.4	92.2	68.1	87.2	93.2	93.7	70.0	92.7
	F1-score (%)	93.1	95.0	67.0	88.5	92.4	95.2	68.3	91.6
	AUC (%)	92.6	95.2	69.2	90.2	93.0	95.5	70.2	93.0
	G-Mean (%)	92.6	95.2	69.2	90.2	93.0	95.4	70.2	93.0
	Pfa (%)	4.2	1.7	29.7	2.7	7.2	2.9	25.8	4.8
ML	Precision (%)	64.2	50.4	66.3	45.2	51.4	55.1	57.4	53.0
	Recall (%)	25.5	64.7	19.2	36.6	32.8	54.4	20.1	36.1
	F1-score (%)	36.5	56.7	29.8	40.5	40.0	54.8	29.8	43.0
	AUC (%)	55.4	56.7	52.2	49.5	53.3	58.5	53.7	54.5
	G-Mean (%)	47.4	50.7	42.0	47.8	49.2	53.3	41.9	51.3
	Pfa (%)	12.0	27.2	13.1	37.6	26.3	37.5	12.6	27.0
Model	#Client	4							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+OS	ALMITY+OS	CTM <sub>client</sub> +OS	CTM <sub>all</sub> +OS
DL	Precision (%)	92.3	98.2	57.2	83.5	93.0	97.2	64.1	91.4
	Recall (%)	91.3	92.4	64.6	95.2	92.0	95.1	60.8	93.9
	F1-score (%)	91.8	95.2	60.7	88.9	92.5	96.1	64.1	91.4
	AUC (%)	92.4	95.4	61.9	89.6	93.1	96.4	68.2	92.6
	G-Mean (%)	92.4	95.4	61.9	90.6	93.1	96.4	68.2	92.6
	Pfa (%)	6.4	1.4	40.7	6.0	5.8	2.3	24.3	1.6
ML	Precision (%)	57.9	58.3	66.1	65.3	55.0	54.3	60.0	56.0
	Recall (%)	44.7	53.0	20.0	39.5	40.0	59.8	18.7	36.3
	F1-score (%)	50.4	55.5	30.8	40.1	46.6	57.0	57.0	44.1
	AUC (%)	58.6	60.5	54.7	55.4	56.7	58.6	54.1	55.5
	G-Mean (%)	56.9	60.0	42.8	48.2	54.2	58.6	40.9	52.5
	Pfa (%)	27.5	22.0	4.0	10.7	26.7	32.5	10.5	24.2
Model	#Client	5							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+OS	ALMITY+OS	CTM <sub>client</sub> +OS	CTM <sub>all</sub> +OS
DL	Precision (%)	93.0	96.9	55.9	90.5	95.1	97.7	60.4	89.9
	Recall (%)	86.1	94.0	76.1	83.1	85.7	93.5	61.5	94.5
	F1-score (%)	89.4	95.4	64.5	86.6	90.2	95.6	60.9	92.1
	AUC (%)	90.3	95.7	62.6	89.0	91.0	95.8	63.7	93.2
	G-Mean (%)	90.2	95.7	61.1	88.8	90.2	95.8	63.6	93.2
	Pfa (%)	5.5	2.6	50.9	5.0	3.7	1.8	34.1	18.1
ML	Precision (%)	61.0	64.3	76.1	59.1	59.5	55.8	66.2	55.0
	Recall (%)	30.7	37.4	13.0	34.6	43.0	58.9	19.6	37.2
	F1-score (%)	40.9	47.3	22.2	41.2	49.9	57.3	30.3	44.4
	AUC (%)	57.1	60.0	51.8	58.1	48.7	59.8	55.5	55.7
	G-Mean (%)	50.6	55.6	35.4	51.1	56.9	59.8	42.4	52.6
	Pfa (%)	16.6	17.5	3.4	27.0	24.7	29.4	8.5	25.8
Model	#Client	6							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+OS	ALMITY+OS	CTM <sub>client</sub> +OS	CTM <sub>all</sub> +OS
DL	Precision (%)	91.2	93.8	68.5	90.2	94.6	96.3	63.2	93.3
	Recall (%)	92.2	94.8	44.2	85.1	92.2	94.3	58.3	92.9
	F1-score (%)	91.7	94.3	53.8	87.6	93.4	95.3	60.7	93.1
	AUC (%)	92.3	94.8	63.5	89.7	93.9	95.6	64.8	94.4
	G-Mean (%)	92.3	94.8	60.5	89.6	93.9	95.6	64.5	94.4
	Pfa (%)	7.4	5.2	17.2	5.6	4.4	3.1	28.6	2.6
ML	Precision (%)	60.9	63.6	66.1	62.7	60.3	56.0	55.4	53.2
	Recall (%)	22.2	33.3	20.0	34.5	36.4	54.1	23.3	51.7
	F1-score (%)	32.6	43.7	30.8	33.4	45.5	55.2	32.8	52.4
	AUC (%)	55.1	58.6	54.7	57.3	57.9	58.1	53.7	56.2
	G-Mean (%)	44.2	52.9	42.8	46.1	53.9	57.9	44.3	56.4
	Pfa (%)	12.0	16.1	8.7	13.9	20.3	30.2	15.9	38.5
DL	Cliff's d	1.0	-	1.0	1.0	1.0	-	1.0	1.0
ML	Cliff's d	0.6	-	1.0	0.84	0.6	-	1.0	0.84

TABLE II  
THE PERFORMANCE OF DIFFERENT TRAINING METHODS IN THE CONTEXT OF DEFECT PREDICTION (DP) TASK

Model	#Client	2							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+Oversampling	ALMITY+Oversampling	CTM <sub>client</sub> +Oversampling	CTM <sub>all</sub> +Oversampling
DL	Precision (%)	60.2	60.6	57.8	58.4	74.1	75.5	59.9	60.3
	Recall (%)	45.8	53.0	38.3	51.9	68.5	71.1	40.8	54.0
	F1-score (%)	52.1	56.6	46.1	55.0	71.2	73.2	48.5	57.1
	AUC (%)	60.2	62.0	57.3	60.3	74.1	75.8	58.8	62.2
	G-Mean (%)	58.4	61.3	54.1	59.8	73.9	75.6	56.0	61.7
	Pfa (%)	29.0	25.5	31.2	25.1	20.2	19.5	26.0	23.6
	CostEffect@5%	36.4	41.4	29.4	38.9	70.3	74.2	42.1	45.1
	Popt@5%	37.2	38.4	38.9	42.6	65.9	70.9	43.1	47.1
	CostEffect@20%	50.8	54.6	38.4	49.3	69.5	71.7	39.2	48.8
	Popt@20%	46.7	51.0	35.1	48.7	69.4	72.0	39.6	46.3
ML	Precision (%)	41.8	53.9	46.7	55.6	44.4	68.3	39.8	41.7
	Recall (%)	35.1	78.7	84.0	91.0	43.2	36.4	86.3	47.7
	F1-score (%)	38.2	64.0	60.0	69.1	43.8	47.5	54.5	44.5
	AUC (%)	53.7	55.7	51.6	59.3	56.2	61.1	56.1	54.9
	G-Mean (%)	50.3	50.8	40.1	50.1	54.7	55.9	47.3	54.5
	Pfa (%)	27.7	67.2	80.9	72.5	30.7	24.2	74.1	37.8
	#Client	3							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+OS	ALMITY+OS	CTM <sub>client</sub> +OS	CTM <sub>all</sub> +OS
DL	Precision (%)	57.8	61.0	60.4	61.7	69.5	70.5	56.5	63.6
	Recall (%)	44.4	57.2	36.0	46.9	57.3	64.9	47.8	47.9
	F1-score (%)	50.2	59.1	45.1	53.3	69.5	70.5	51.8	54.7
	AUC (%)	58.5	63.2	58.1	61.2	68.0	70.9	58.3	62.9
	G-Mean (%)	56.8	62.9	53.7	59.5	67.2	70.7	57.4	59.5
	Pfa (%)	30.9	24.5	31.1	27.5	23.0	19.9	24.6	21.2
	CostEffect@5%	35.9	60.0	28.1	54.7	58.7	66.3	39.0	45.3
	Popt@5%	38.0	60.5	27.1	54.2	58.0	65.9	41.4	46.0
	CostEffect@20%	60.9	55.2	32.2	44.0	58.2	65.0	38.3	47.2
	Popt@20%	58.5	56.7	31.1	41.0	58.4	65.1	38.6	44.3
ML	Precision (%)	68.7	64.1	46.6	40.1	38.3	42.1	40.1	39.4
	Recall (%)	29.3	33.3	82.7	81.7	54.2	80.0	86.0	51.0
	F1-score (%)	41.1	43.9	59.6	53.7	44.8	55.1	54.6	44.5
	AUC (%)	59.0	58.8	51.4	56.2	52.3	58.8	56.6	53.3
	G-Mean (%)	51.0	53.0	40.9	50.1	52.2	54.8	48.3	53.2
	Pfa (%)	11.2	15.7	79.8	69.2	49.6	62.3	65.1	44.4
	#Client	4							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+OS	ALMITY+OS	CTM <sub>client</sub> +OS	CTM <sub>all</sub> +OS
DL	Precision (%)	57.3	58.4	56.3	50.4	63.4	66.9	57.2	62.0
	Recall (%)	40.2	56.4	35.6	62.0	59.7	60.8	39.7	50.4
	F1-score (%)	47.2	57.4	43.6	55.6	61.5	63.7	46.9	55.6
	AUC (%)	57.5	62.2	56.1	61.3	65.3	67.7	57.3	63.4
	G-Mean (%)	54.8	61.1	52.2	61.0	65.1	67.3	54.5	62.8
	Pfa (%)	30.8	23.3	29.0	25.2	25.4	23.4	29.1	25.1
	CostEffect@5%	35.4	55.8	33.2	52.2	31.1	59.2	34.5	52.7
	Popt@5%	36.6	55.5	33.5	61.4	32.7	60.6	34.5	53.3
	CostEffect@20%	33.8	54.4	34.9	52.6	34.8	58.7	34.6	49.6
	Popt@20%	34.4	55.0	36.7	51.1	33.8	59.4	34.3	49.6
ML	Precision (%)	41.6	51.6	39.5	45.5	66.0	62.2	54.5	73.0
	Recall (%)	85.9	33.8	90.1	39.4	35.0	41.8	89.6	34.5
	F1-score (%)	56.1	40.9	54.9	42.3	45.8	50.0	67.7	46.9
	AUC (%)	58.9	57.9	56.0	56.4	58.5	60.3	57.4	62.0
	G-Mean (%)	52.3	52.7	44.4	53.7	53.6	57.4	47.4	55.6
	Pfa (%)	68.1	17.9	78.1	26.7	18.0	21.2	74.9	10.6
	#Client	5							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+OS	ALMITY+OS	CTM <sub>client</sub> +OS	CTM <sub>all</sub> +OS
DL	Precision (%)	55.3	56.9	54.4	60.5	65.6	64.3	54.6	61.7
	Recall (%)	44.3	52.8	26.4	47.7	46.4	56.9	41.5	49.6
	F1-score (%)	49.2	54.8	35.6	53.3	64.3	65.6	47.2	55.4
	AUC (%)	57.0	59.5	53.8	58.5	62.9	65.1	56.1	60.7
	G-Mean (%)	55.6	59.1	46.3	57.3	60.7	64.5	54.2	59.3
	Pfa (%)	28.9	18.7	30.3	26.3	26.7	20.6	29.3	26.4
	CostEffect@5%	45.3	52.6	21.4	51.2	40.0	52.5	21.0	47.2
	Popt@5%	45.2	53.3	19.1	51.2	39.6	51.0	19.5	47.4
	CostEffect@20%	36.6	46.2	33.3	47.1	43.3	54.3	23.4	42.1
	Popt@20%	38.8	49.4	29.8	46.2	41.6	53.5	22.3	41.6
ML	Precision (%)	47.9	50.2	46.1	40.2	44.7	45.8	58.3	45.9
	Recall (%)	30.7	33.7	79.5	84.7	86.0	58.8	24.6	69.3
	F1-score (%)	37.4	40.3	58.3	54.5	58.9	51.5	34.6	55.2
	AUC (%)	55.9	57.4	51.1	56.6	62.9	59.7	57.3	61.5
	G-Mean (%)	49.9	51.3	42.4	49.2	58.5	59.7	47.0	60.0
	Pfa (%)	18.9	18.9	77.3	71.4	60.2	39.3	10.0	46.3
	#Client	6							
	Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+OS	ALMITY+OS	CTM <sub>client</sub> +OS	CTM <sub>all</sub> +OS
DL	Precision (%)	54.1	60.6	61.3	57.9	58.5	62.1	69.5	60.2
	Recall (%)	37.5	45.5	32.4	43.2	45.3	53.5	33.4	51.9
	F1-score (%)	44.3	51.9	42.4	49.6	51.1	57.5	45.1	55.7
	AUC (%)	55.4	60.3	57.6	59.4	59.1	62.9	60.5	61.4
	G-Mean (%)	52.4	58.4	51.8	54.8	57.4	62.2	59.3	60.7
	Pfa (%)	27.7	17.2	29.0	24.9	26.8	12.4	27.2	25.6
	CostEffect@5%	16.1	38.0	32.4	50.1	46.8	47.6	20.9	47.1
	Popt@5%	15.4	40.9	38.7	51.6	47.2	47.3	20.2	47.7
	CostEffect@20%	27.7	46.4	26.0	45.6	45.7	47.9	21.9	40.6
	Popt@20%	25.4	44.3	29.3	46.5	45.5	47.1	21.5	41.5
ML	Precision (%)	38.8	47.1	71.4	36.6	41.5	41.7	46.2	40.2
	Recall (%)	72.3	34.0	18.2	52.1	44.6	80.9	25.5	46.1
	F1-score (%)	50.6	39.5	29.0	43.0	43.1	55.1	32.9	42.9
	AUC (%)	54.1	56.2	56.1	50.6	54.7	58.7	54.4	53.7
	G-Mean (%)	51.0	51.7	41.3	50.6	53.8	54.3	46.1	53.2
	Pfa (%)	64.1	21.6	18.2	50.8	35.3	63.5	16.8	38.6
DL	Cliff's d	0.96	-	1.0	0.52	0.36	-	1.0	0.76
ML	Cliff's d	0.60	-	1.0	0.60	0.92	-	1.0	0.36

**Metrics.** In addressing this RQ, we employ a standard set of widely recognized evaluation metrics to assess the performance of various training methods. These metrics encompass Precision, Recall, F1-score, G-Mean, AUC, and the Probability of False Alarm (Pfa). We exclude the metric, Accuracy, from our evaluation criteria because it can introduce a large bias when assessing model performance on skewed datasets. For the DL defect prediction task, we augment our evaluation with two additional metrics: two effect-aware metrics, i.e., CostEffect [49] and Popt [49], [50], which are utilized for conducting cost-benefit analysis. We assess these two effort-aware metrics at specific levels: *CostEffect@5%*, *CostEffect@20%*, *Popt@5%*, and *Popt@20%*. Please be aware that each data instance in the dataset used for training the ML-based defect predictor comprises 22 different metrics, with no corresponding source code available. Therefore, the application of effort-aware metrics on this dataset is not suitable. With this in mind, we exclusively conduct a cost-benefit analysis on the DL-based defect predictor using these two metrics. Moreover, we conduct an effect size analysis, utilizing Cliff's delta [50], [51], [52], a widely recognized metric in effect size analysis, to facilitate the interpretation of the significance of our experimental results. Specifically, we employ G-Mean to assess the performance of various training methods and calculate the effect size. G-Mean is chosen for this purpose because it exhibits the least susceptibility to bias from the class imbalance among performance metrics [53]. The resulting value of Cliff's delta can be interpreted as follows [53]: 1) A no-effect size: Cliff's delta is smaller than 0.11. 2) A small effect size: Cliff's delta is  $\in [0.11, 0.28)$ . 3) A medium effect size: Cliff's delta is  $\in [0.28, 0.43)$ . 4) A large effect size: Cliff's delta is no less than 0.43. In our study, a large Cliff's delta indicates that ALMITY has a positive impact on training well-performed models, and each effect size value in the tables represents the effect size between ALMITY and a respective baseline method.

**Results.** The experimental results are presented in Table I and Table II. Based on these experimental findings, we can draw the following observations:

- 1) **In the context of academic dataset distributions, ALMITY consistently outperforms the baseline methods in a set of evaluation aspects, whether applied to training DL or ML models in different SE tasks.** By examining Table I and Table II, it is evident that ALMITY consistently outperforms the baseline methods in terms of the G-Mean measure, regardless of the number of clients involved in the training process, the type of learning-based models (i.e., ML and DL), and the specific SE task under consideration. As an illustration, in the PD task, ALMITY attains a G-Mean score of approximately 60%, surpassing VFL and CTM by 3-4% while ALMITY also exhibits a similar trend in the CCD task. In the majority of instances, Cliff's d values for ALMITY indicate a substantial enhancement in performance on academic datasets with mildly imbalanced data distributions. Only in two instances within the DP task does ALMITY exhibit a moderate impact on performance improvement compared to baseline methods. Regarding the cost-effectiveness aspect, in the majority of cases, ALMITY exhibits relatively higher CostEffect and Popt values in training

the DL-based defect predictor compared to the baseline methods, especially for VFL. Furthermore, ALMITY consistently attains the lowest Pfa values across the majority of cases. Even in instances where ALMITY exhibits a relatively higher Pfa value, the Precision and Recall metrics for our framework outperform those of other training methods. Regarding the Cliff's delta metric, 22 out of 24 Cliff's delta values exceed 0.43, illustrating the important significance of ALMITY's performance improvement. In contrast, only two instances yield a value of 0.36, indicating a moderate level of significance in ALMITY's performance improvement. These findings provide evidence supporting ALMITY's effectiveness in training high-performing models on academic datasets.

- 2) **ALMITY can enhance the performance of learning-based models trained on multiple small-scale datasets compared to models trained on the entire dataset.** In the majority of cases for both DP and CCD tasks, ALMITY outperforms the performance of CTM on the entire dataset. A thorough analysis suggests that this superiority may be attributed to the inherent class imbalance in academic datasets, especially within the DP datasets, where the ratio of defective instances to non-defective instances is nearly 1:3. Our method, ALMITY, facilitates the acquisition of more suitable model parameters in fewer training iterations, thanks to our parameter aggregation strategy. Consequently, with an equivalent number of training rounds, ALMITY's parameter aggregation strategy generates more optimal model parameters and thus empowers learning-based models to achieve higher performance compared to training models on the complete dataset.
- 3) **The application of oversampling techniques results in performance improvements of varying magnitudes across different training methods.** The implementation of oversampling techniques proves beneficial for enhancing the performance of various categories of learning-based models in both CCD and DP tasks. For example, in the CCD task, the use of oversampling elevates the G-Mean measure for DL models from approximately 95.0 to well over 96.0, while it contributes to a G-Mean improvement of around 3-4% for ML models. Moreover, the utilization of oversampling techniques can assist ALMITY and VFL in surpassing the performance of the CTM method across the majority of cases, resulting in improvements. For example, in the DP task, the application of the oversampling technique elevates ALMITY's performance from 61.3% to 75.6% in terms of G-Mean and enhances VFL's performance from 58.4% to 73.9% when utilizing just 2 clients. This improvement far surpasses the performance enhancement achieved by CTM. This is attributed to the fact that ALMITY and VFL involve multiple clients, necessitating oversampling within each client's dataset. As a result of this process, the oversampled datasets encompassing all clients in ALMITY and VFL are to some extent larger than the dataset after implementing the oversampling technique employed in CTM. Note that this phenomenon is not an experimental error. It arises from the inherent distinctions between the

FL-based framework and CTM. Overall, in the majority of instances, ALMITY outperforms CTM and VFL after implementing the oversampling technique. Moreover, in many cases, our framework without the oversampling technique still surpasses CTM when this technique is employed.

- 4) **For FL-based training methods, i.e., VFL and ALMITY, their ultimate performance remains unaffected by the number of clients they encompass.** Drawing insights from Table I and Table II, it becomes evident that in the context of FL-based training methods, namely VFL and ALMITY, the performance of the model in different tasks remains consistently stable within a certain range, regardless of the number of clients. As an example, when considering ALMITY, in the CCD task, the DL model consistently achieves approximately 95% ( $\pm 1\%$ ) in G-Mean, while the ML model maintains an approximately 55% ( $\pm 5\%$ ) performance level. In the DP task, the DL model tends to hover around 60% ( $\pm 3\%$ ), while the ML model maintains a performance of roughly 50% ( $\pm 5\%$ ).
- 5) **At least two factors can influence the performance of a learning-based model: the complexity of the task and the nature of the input.** In the realm of DL models, training a highly effective defect predictor is a more challenging task compared to training a clone detector. Drawing upon the insights gleaned from the aforementioned findings, it becomes evident that, on the datasets with similar data scales and data balance, the DL-based clone detector consistently attains a G-Mean score exceeding 95%, a substantial improvement of around 35 percent points compared to the DL-based defect predictor's performance. This phenomenon arises from the inherent advantage of DL models in the task of generating code representation and calculating the similarity between the representations of these two code fragments, as opposed to the more complex task of identifying bugs within code fragments based on their code representations. Hence, this disparity in performance can be attributed to the inherent complexity of the tasks involved. In addition, in the CCD, a noteworthy distinction arises: the ML-based clone detector gets worse performance than the DL-based detector. The ML model encounters challenges in comprehending the intricacies of code representations generated by CodeGraphBERT. Therefore, the DL-based detector predictor outperforms the ML model. This illustrates that ML-based models are better suited for processing metric-based data as opposed to complex representations.

🔗 **RQ-1** ▶ *In this RQ, we substantiate the potential and feasibility of employing FL-based approaches, namely VFL and ALMITY, for addressing SE tasks. Furthermore, on academic datasets, we discern that ALMITY consistently outperforms baseline methods across various model types and diverse SE tasks.* ◀

*B. RQ-2: Can ALMITY Outperform Baselines on Diverse Skewed Data Distributions?*

**Motivation.** Real-world industrial datasets often exhibit varying scales (specific to each company) and levels of data balance (task-dependent). However, employing conventional training methods like CTM and FL to train deep learning models on imbalanced datasets may lead to a decline in performance. To address this challenge, we introduce a novel framework, ALMITY, designed to enhance model performance on skewed datasets while preserving data privacy. In this RQ, we aim to assess the effectiveness of ALMITY on datasets with diverse data distributions. Our comprehensive experiments evaluate the performance of different training methods, seeking to determine whether ALMITY can indeed improve model performance on skewed datasets, thereby addressing an important concern in industrial settings.

**Method.** As previously discussed in the background, two prevalent types of skews found in datasets are quantity skew and label distribution skew. Consequently, in our assessment of ALMITY's effectiveness, we systematically manipulate the attributes related to data scale and data balance. This enables us to generate a diverse set of skewed datasets with varying data distributions, allowing for a comprehensive evaluation of ALMITY's performance across these diverse scenarios.

In order to create a variety of imbalanced datasets that encompass a wide spectrum of data distributions, and to ensure that these generated datasets accurately mirror the task-specific distributions encountered in real-world scenarios for both the CCD and DP tasks, we categorize nine distinct types of data distributions. These data distribution classifications are derived from adjustments in data scale and data balance, i.e.,  $\Phi$  and  $\alpha$ . To begin, we conduct an initial examination of the real-world distribution of datasets pertaining to these two tasks. For the distribution of the real-world CCD dataset, several prior studies [17], [18] have indicated that software systems commonly exhibit approximately 10% to 20% cloned code, even with cloned instances of proportions exceeding 50%. Additionally, for the DP task, Ni et al. [36] noted that the defect ratios in different projects vary, ranging from 15% to 30%. Hence, drawing upon the insights gained from the previous analysis, we establish two thresholds among real-world task-specific data distribution to categorize the data scale ( $\Phi$ ) and data balance ( $\alpha$ ) of a dataset into three discernible levels: low-range, medium-range, and high-range values, respectively. More specifically, we select a significant value, 50%, among task-specific data distributions as the threshold for distinguishing between high-range and medium-range attribute values. Additionally, we utilize a value, i.e., 20%, which is present in both the CCD and DP task data distributions, as another threshold to differentiate between low-range and medium-range values. Expanding on this concept, for data scale: 1) The data scale attribute is categorized as high-range if the number of its data instances exceeds 50% of the optimal data scale. 2) The data scale attribute is considered medium-range if the number of its data instances falls within the range of 50% to 20% of the optimal data scale. 3) The data scale attribute is classified as low-range if the number of its data

instances is less than 20% of the optimal data scale. For the data balance attribute: 1) The data balance attribute is categorized as high-range when the ratio of negatives to positives surpasses 1:2. 2) The data balance attribute is considered medium-range if the ratio of negatives to positives falls within the range of 1:2 to 1:5. 3) The data balance attribute is classified as low-range if the ratio of negatives to positives is less than 1:5. To facilitate a clearer understanding, we offer a comprehensive example for illustration. Let us consider an FL-based framework comprising four clients, with a total of 400 data instances distributed among them. In this scenario, the optimal values for data scale and data balance in each client are set at 100 and 1:1 (50 positives and negatives), respectively. Suppose a client possesses 60 data instances, with 10 of them being positive. In this case, the client is characterized by a high-range value in terms of data scale (representing 60% of the optimal data scale value) and a low-range value with regard to data balance (constituting only 20% of the optimal data balance value,  $10 : (60 - 10)$ ).

Currently, the classification for data scale attribute values is as follows: High-range value (H): When  $\Phi \geq 0.5$ ; Medium-range value (M): When  $0.2 < \Phi < 0.5$ ; Low-range value (L): When  $\Phi \leq 0.2$ . Similarly, the classification for data balance attribute values is as follows: High-range value (H): When  $\alpha \geq 1 : 2$ ; Medium-range value (M): When  $1 : 5 < \alpha < 1 : 2$ ; Low-range value (L): When  $\alpha \leq 1 : 5$ . Next, we amalgamate the three distinct classifications for each attribute, resulting in a total of nine different data distributions: 1) *HH: high-range data scale and high-range data balance*, 2) *HM: high-range data scale and medium-range data balance*, 3) *HL: high-range data scale and low-range data balance*, 4) *MH: medium-range data scale and high-range data balance*, 5) *ML: medium-range data scale and medium-range data balance*, 6) *MM: medium-range data scale and low-range data balance*, 7) *LH: low-range data scale and high-range data balance*, 8) *LM: low-range data scale and medium-range data balance*, and 9) *LL: low-range data scale and low-range data balance*. In the end, we produce a total of 36 datasets, with each set of four datasets belonging to one of the data distribution types mentioned earlier.

**Experiment 1:** To ascertain the effectiveness and practicality of ALMITY across various data distributions, we carry out experiments aimed at comparing the performance of different training methods within these distinct data distribution contexts. We configure the number of clients for both FL and ALMITY to be four, which corresponds to the median value observed in RQ1. In our pursuit of replicating real-life conditions as closely as possible within our experimental settings, we rigorously maintain the independence of the various client-side datasets. Consequently, when assessing the effectiveness of CTM in each unique combination, we determine the average performance across all four clients as the final result.

**Experiment 2:** In practical real-world scenarios, FL-based frameworks often encounter diverse data distributions. Bearing this in mind, we incorporate multiple datasets, each representing various data distributions, within the same FL-based framework. Specifically, we create both a VFL and the ALMITY framework, each comprising nine distinct clients. Each client

is equipped with a dataset characterized by a unique type of data distribution. Similar to the approach taken in RQ1, we still consider CTM as a baseline method. In the case of using CTM, we assess the model's performance on each individual client-side dataset, as well as its performance when trained on the comprehensive dataset resulting from the integration of nine distinct datasets.

**Metrics.** Likewise, in this RQ, we employ the identical set of evaluation metrics as utilized in RQ1. These metrics encompass Precision, Recall, F1-score, AUC, G-Mean, and the Probability of False Alarm (Pfa) to assess the model's performance. Additionally, we examine cost-effectiveness through the metrics CostEffect and Popt at 5% and 20% within the code churn. To validate improvements in ALMITY's effectiveness, we also incorporate Cliff's delta as an effect size metric.

**Statistical Test.** To ensure the accuracy of ALMITY, we employ a statistical test to validate the significance of its performance across various data distributions. We utilize the widely recognized Wilcoxon test [54] in our study, a statistical test that has been employed in numerous prior studies [55], [56]. In our analysis, we conduct the Wilcoxon test, comparing ALMITY's performance on different types of data distributions with each baseline method to calculate the respective p-values. Following the experimental methodology outlined in [55], [56], we evaluate the p-value generated by the test at the commonly accepted significance level of 0.05. Note that we employ the G-Mean as the metric for the statistical test, as the G-Mean serves as a summary evaluation metric unaffected by skewed data distributions. Our null and alternative hypotheses are as follows:

- 1)  $H_0$ : ALMITY cannot achieve better performance than baseline methods.
- 2)  $H_1$ : ALMITY would tend to achieve better performance than baseline methods.

**Results.** The experimental results are presented in Table III to Table VII. According to the results, we can obtain the following observations:

- 1) **Experiment 1: Compared with baseline methods, ALMITY can enhance the performance of the learning-based model across various data distributions.** From the preceding four tables, we observe that among the nine different types of data distributions, ALMITY consistently outperforms other training methods. Additionally, based on Cliff's d values comparing ALMITY's G-Mean performance with that of the baselines across these nine data distributions, 49 out of the total 72 Cliff's d values indicate that ALMITY exhibits a statistically significant improvement in model performance compared to the baselines. Out of these Cliff's delta values, 15.3% (11) indicate that ALMITY improves the performance of learning-based models to a moderate extent, while another 15.3% (11) show a small improvement. Notably, in the context of the CCD task, there is only one instance where ALMITY achieves the same performance as CTM on the entire dataset in the ML distribution, and this occurs after employing the oversampling technique. This is attributed to the application of the oversampling

TABLE III  
THE PERFORMANCE OF VARIOUS TRAINING METHODS ON DATASETS WITH HIGH SCALE AND DIVERSE DATA BALANCE DISTRIBUTIONS

Task	Model	Distribution	HH							
		Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+Oversampling	ALMITY+Oversampling	CTM <sub>client</sub> +Oversampling	CTM <sub>all</sub> +Oversampling
CCD	DL	Precision (%)	95.3	96.3	60.0	92.6	86.5	91.7	77.4	95.4
		Recall (%)	92.2	93.8	90.8	94.6	84.5	92.6	51.3	94.5
		F1-score (%)	93.7	95.0	71.8	93.6	85.5	92.1	60.5	91.8
		AUC (%)	94.1	95.6	66.7	94.1	86.2	92.5	75.1	95.2
		G-Mean (%)	94.1	95.6	61.3	94.1	86.2	92.5	69.7	94.9
		Pfa (%)	4.1	2.5	47.3	4.5	12.0	7.6	21.1	4.1
	ML	Precision (%)	47.7	49.2	68.3	55.0	45.8	54.6	58.4	46.7
		Recall (%)	46.4	51.8	22.7	37.2	45.0	40.9	22.9	56.3
		F1-score (%)	47.1	50.4	34.0	44.4	45.4	46.8	32.9	51.1
		AUC (%)	53.0	56.8	57.1	55.7	50.0	56.1	56.9	51.0
		G-Mean (%)	52.6	56.6	45.6	52.6	49.8	52.9	44.4	50.7
		Pfa (%)	40.4	38.1	8.4	25.8	44.9	28.8	13.8	54.3
DP	DL	Precision (%)	52.8	59.0	54.7	62.7	66.5	69.1	58.8	62.3
		Recall (%)	37.6	51.7	18.0	49.0	46.0	54.8	48.7	44.6
		F1-score (%)	43.9	55.1	27.1	55.0	54.4	61.2	53.7	52.0
		AUC (%)	56.8	63.0	53.3	62.7	62.5	67.6	60.1	61.4
		G-Mean (%)	53.5	62.0	40.0	61.1	60.3	66.4	59.0	59.0
		Pfa (%)	24.0	25.6	11.4	23.6	21.0	19.6	28.5	21.8
	ML	CostEffect@5% (%)	32.5	37.5	17.4	36.4	46.5	57.4	28.3	43.2
		Popt@5% (%)	36.6	42.5	17.1	41.5	45.6	56.4	28.1	50.7
		CostEffect@20% (%)	32.4	39.1	24.1	38.7	46.4	54.4	30.2	45.1
		Popt@20% (%)	32.1	37.9	21.8	37.0	47.5	56.1	29.1	45.5
		Precision (%)	68.3	59.9	45.2	49.0	52.6	41.3	51.7	39.1
		Recall (%)	36.4	56.5	88.7	53.0	86.0	68.6	68.1	92.0
CCD	DL	Precision (%)	89.4	92.1	0.0	92.1	87.3	89.4	48.8	74.8
		Recall (%)	86.9	86.9	0.0	70.2	85.0	91.6	64.1	84.9
		F1-score (%)	88.2	89.4	0.0	82.8	86.1	91.6	55.4	83.5
		AUC (%)	91.4	92.0	0.0	89.5	90.4	94.1	72.5	93.8
		G-Mean (%)	91.3	91.8	0.0	87.1	90.3	94.1	72.0	92.8
		Pfa (%)	4.0	2.9	0.0	12.8	4.1	3.3	19.2	8.3
	ML	Precision (%)	60.6	44.6	66.2	59.7	45.2	51.4	46.5	48.3
		Recall (%)	24.2	38.0	13.3	24.6	36.6	32.8	81.1	79.0
		F1-score (%)	34.5	41.0	22.1	34.9	40.5	40.0	59.1	60.0
		AUC (%)	55.4	49.0	53.7	55.3	49.5	53.3	51.1	53.7
		G-Mean (%)	45.8	47.8	35.4	46.0	47.8	49.2	41.4	47.4
		Pfa (%)	13.3	39.9	5.8	14.1	37.6	26.3	78.9	71.5
DP	DL	Precision (%)	60.0	64.7	0.0	68.8	61.5	83.7	72.9	61.1
		Recall (%)	3.6	10.6	0.0	7.7	45.3	59.5	42.1	41.0
		F1-score (%)	6.7	18.2	0.0	13.9	52.2	69.6	53.4	49.1
		AUC (%)	51.4	54.3	0.0	53.2	61.3	75.1	64.8	60.1
		G-Mean (%)	18.8	32.2	0.0	27.6	59.2	73.5	60.7	57.0
		Pfa (%)	0.7	1.9	0.0	1.3	22.7	9.3	12.5	20.9
	ML	CostEffect@5% (%)	8.3	11.9	0.0	13.7	37.4	57.9	50.0	31.6
		Popt@5% (%)	2.3	8.1	0.0	12.2	37.2	55.8	50.8	33.6
		CostEffect@20% (%)	2.3	9.1	0.0	13.9	39.3	58.3	46.6	39.9
		Popt@20% (%)	2.4	8.9	0.0	13.4	39.9	58.3	48.3	37.9
		Precision (%)	24.6	34.1	33.8	27.8	49.5	48.3	40.0	48.9
		Recall (%)	72.1	77.5	83.9	59.5	93.7	79.0	92.6	93.7
CCD	DL	Precision (%)	95.1	98.6	0.0	79.0	82.3	84.9	80.0	68.0
		Recall (%)	76.3	83.3	0.0	72.6	87.5	92.9	37.6	82.8
		F1-score (%)	84.7	90.6	0.0	75.6	84.8	88.8	51.2	74.6
		AUC (%)	87.7	91.8	0.0	86.0	91.4	94.4	67.7	89.8
		G-Mean (%)	86.9	91.4	0.0	84.8	91.3	94.4	60.6	89.5
		Pfa (%)	1.0	0.3	0.0	10.5	4.6	4.1	2.3	3.1
	ML	Precision (%)	0.0	0.0	0.0	0.0	0.0	75.0	0.0	0.0
		Recall (%)	0.0	0.0	0.0	0.0	0.0	3.3	0.0	0.0
		F1-score (%)	0.0	0.0	0.0	0.0	0.0	6.4	0.0	0.0
		AUC (%)	0.0	0.0	0.0	0.0	0.0	51.4	0.0	0.0
		G-Mean (%)	0.0	0.0	0.0	0.0	0.0	18.2	0.0	0.0
		Pfa (%)	0.0	0.0	0.0	0.0	0.0	0.6	0.0	0.0
DP	DL	Precision (%)	0.0	64.3	0.0	0.0	74.1	92.8	52.1	86.5
		Recall (%)	0.0	3.2	0.0	0.0	43.5	62.0	43.1	54.8
		F1-score (%)	0.0	6.4	0.0	0.0	54.9	74.3	47.2	67.1
		AUC (%)	0.0	50.6	0.0	0.0	65.7	79.1	55.7	74.0
		G-Mean (%)	0.0	16.7	0.0	0.0	61.8	77.2	54.3	71.5
		Pfa (%)	0.0	2.2	0.0	0.0	12.1	3.8	3.2	6.8
	ML	CostEffect@5% (%)	0.0	6.3	0.0	0.0	44.6	60.2	38.4	30.8
		Popt@5% (%)	0.0	2.4	0.0	0.0	44.9	59.1	39.0	28.2
		CostEffect@20% (%)	0.0	2.3	0.0	0.0	49.8	62.1	36.6	23.6
		Popt@20% (%)	0.0	2.7	0.0	0.0	48.6	61.6	37.1	24.1
		Precision (%)	30.0	16.9	30.9	25.0	47.0	54.9	38.4	39.1
		Recall (%)	12.0	19.6	11.0	14.2	90.8	80.1	96.0	92.0
CCD	DL	Precision (%)	0.56	—	1.0	0.56	1.0	—	1.0	0.11
		Recall (%)	0.22	—	0.44	0.22	0.34	—	0.56	0.34
		F1-score (%)	0.34	—	0.56	0.34	1.0	—	1.0	0.78
		AUC (%)	0.34	—	0.56	0.56	0.56	—	1.0	1.0
		G-Mean (%)	0.34	—	0.56	0.56	0.56	—	1.0	1.0
		Pfa (%)	24.0	7.0	6.1	17.6	65.7	70.0	71.5	66.3
	ML	Precision (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		Recall (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		F1-score (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		AUC (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		G-Mean (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		Pfa (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CCD	DL	Chiff's d	0.56	—	1.0	0.56	1.0	—	1.0	0.11
	ML	Chiff's d	0.22	—	0.44	0.22	0.34	—	0.56	0.34
DP	DL	Chiff's d	0.34	—	0.56	0.34	1.0	—	1.0	0.78
	ML	Chiff's d	0.34	—	0.56	0.56	0.56	—	1.0	1.0

technique, which balances the entire dataset by adding minority class data instances, consequently resulting in an improved performance by CTM. Indeed, employing CTM to train a learning-based model on the complete dataset is impractical for real-world application scenarios. This is primarily because, in practice, datasets from various organizations or companies cannot be seamlessly integrated due to concerns about data security and data privacy. Hence, according to the aforementioned analysis, ALMITY consistently excels over the baselines in training high-performing learning-based models across diverse data distributions.

- 2) **Experiment 1: ALMITY enhances the learning capacity of learning-based models when confronted with highly skewed datasets, empowering these models to make effective decisions in such challenging conditions.** For example, in the context of the DP task, ALMITY distinguishes itself from other training methods that fail to enable the DL-based defect predictor to predict any defects within the dataset characterized by the HL data distribution. ALMITY empowers this defect predictor to effectively discern a limited number of genuine defects within datasets of this distribution. This demonstrates that ALMITY enables the DL-based defect predictor to effectively distinguish between defective and non-defective code within such skewed data distributions. Furthermore, a comparable pattern is observed in the CCD task. Following the application of the oversampling technique, ALMITY enables the ML-based clone detector to initiate the identification of certain clones within the dataset characterized by the HL data distribution, achieving a recall rate of 3.3%. In contrast, employing baseline training methods to train the same ML-based clone detector yields no clone identification from datasets of this distribution.
- 3) **Experiment 1: The ML-based defect predictor can outperform the ML-based clone detector, despite the dataset used for training the ML-based defect predictor being smaller and exhibiting more imbalance compared to the dataset used for training the ML-based clone detector.** By referring to Table IV and Table V, it becomes evident that in heavily skewed data distributions, the ML-based defect predictor consistently attains superior performance compared to the ML-based clone detector, despite both models being trained using the same ML algorithm, i.e., SVM. As an example, when examining the MM data distribution, the ML-based defect predictor achieves an approximate 50% G-Mean performance, which is nearly twice as high as that of the clone detector. Besides, in the case of the ML data distribution, the ML-based clone detector fails to attain a satisfactory model performance in clone detection, while the defect predictor still manages to identify defects within this data distribution, scoring approximately 45% in G-Mean. The aforementioned observations highlight that training a highly effective ML model is more achievable when working with straightforward metric-based datasets as opposed to complex high-dimensional vectors.

- 4) **Experiment 1: In certain highly skewed and small-scale data distributions, baseline training methods cannot enable learning-based models to make accurate predictions.** Based on these experimental result tables, for both the CCD and DP tasks, we can find that all training methods cannot train a usable model from the datasets with two types of data distributions, including the ML data distribution and the LL data distribution. The poor performance of ML models on the ML data distribution can be attributed to the low-range values in the data balance attribute. This deficiency results in an insufficient presence of minority class data instances, i.e., defect instances or clones within the dataset characterized by this distribution. Hence, learning-based models struggle to capture the features of the minority class and face challenges in distinguishing between data instances belonging to different classes. This further underscores the significance of the minority class learnability attribute that we introduced during the parameter aggregation process. In addition to the previously mentioned factor, namely the scarcity of minority class instances, another potential contributing factor to the poor performance of both DL and ML models on the LL data distribution is the insufficient quantity of data instances within the dataset. This shortage may hinder models from effectively acquiring knowledge. Except for the ML and LL data distributions, in the DP task, ALMITY achieves slightly higher performance than other baselines in the HL and MM distributions. The baseline training methods cannot identify any defects from datasets, while ALMITY can identify a small proportion of defects correctly. The fundamental reasons are similar to those mentioned earlier. For the HL data distribution, despite the large number of data instances in this distribution, only a few instances are real defects, making it challenging for the DL-based defect predictor to learn to distinguish defects from non-defective code. The poor performance within the MM data distribution can be attributed to the insufficient number of medium-scale data instances, which makes it challenging for DL models to acquire meaningful knowledge from this dataset. However, a slight improvement in ALMITY's performance indicates that our framework can effectively leverage fewer data instances or more imbalanced datasets to capture distinctions between defective and non-defective code.
- 5) **Experiment 1: Across real-world task-specific data distributions, ALMITY consistently proves effective in enhancing the performance of learning-based models.** Given that the classification of various data distributions is grounded in real-world, task-specific data scenarios, among the nine data distributions considered, the HL, ML, and LL distributions closely approximate real-world defect and clone distributions at varying scales of datasets. Our experimental results on these three distributions consistently demonstrate ALMITY's superiority over the baselines in training learning-based models with higher G-Mean performance.



TABLE IV  
THE PERFORMANCE OF VARIOUS TRAINING METHODS ON DATASETS WITH MEDIUM SCALE AND DIVERSE DATA BALANCE DISTRIBUTIONS

Task	Model	Distribution	MH									
		Method	VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+Oversampling	ALMITY+Oversampling	CTM <sub>client</sub> +Oversampling	CTM <sub>all</sub> +Oversampling		
CCD	DL	Precision (%)	92.0	93.7	67.4	84.9	94.7	95.4	72.4	90.7		
		Recall (%)	83.1	90.6	69.2	85.9	90.4	92.4	72.4	77.5		
		F1-score (%)	87.3	92.1	68.3	85.4	92.5	93.9	72.4	80.5		
		AUC (%)	88.2	92.8	72.7	87.6	93.3	94.5	75.5	85.6		
		G-Mean (%)	88.1	92.7	72.6	87.6	93.2	94.4	75.5	85.5		
		Pfa (%)	6.6	5.1	33.8	15.6	3.9	3.4	31.4	14.3		
	ML	Precision (%)	55.5	56.9	52.9	49.6	48.7	46.5	59.4	52.2		
		Recall (%)	25.0	33.0	18.0	27.5	73.3	45.3	18.3	35.9		
		F1-score (%)	34.5	41.8	26.9	35.4	58.6	45.9	28.0	45.1		
		AUC (%)	55.4	56.5	53.3	53.8	54.3	51.8	54.2	54.3		
		G-Mean (%)	46.3	51.4	39.9	46.9	50.9	51.6	40.6	51.7		
		Pfa (%)	14.3	20.0	11.4	20.0	64.6	35.7	10.0	35.5		
DP	DL	Precision (%)	61.9	66.7	68.0	58.9	57.6	64.1	61.4	59.3		
		Recall (%)	33.3	46.2	10.9	45.3	51.0	58.3	49.6	47.7		
		F1-score (%)	43.3	54.5	18.8	51.2	54.1	61.1	54.9	52.4		
		AUC (%)	59.4	64.9	53.6	59.8	60.5	66.1	62.3	60.3		
		G-Mean (%)	53.4	62.1	32.4	58.0	59.7	65.6	61.0	58.4		
		Pfa (%)	14.6	16.4	3.7	25.6	30.0	26.1	24.9	26.0		
		CostEffect@5% (%)	30.5	54.3	39.9	40.5	52.5	56.9	39.4	50.0		
		Popt@5% (%)	27.4	52.9	27.8	47.3	57.9	60.2	39.4	53.6		
		CostEffect@20% (%)	33.0	45.8	38.3	40.7	50.2	53.9	40.4	44.4		
	Popt@20% (%)	32.0	42.0	39.2	40.1	50.8	52.2	39.6	47.3			
	ML	Precision (%)	46.7	48.2	29.2	47.4	43.4	45.3	33.3	48.6		
		Recall (%)	64.1	45.0	95.2	55.6	44.2	57.5	27.9	85.3		
		F1-score (%)	54.0	46.6	44.6	51.2	43.8	50.7	30.4	61.9		
		AUC (%)	51.7	55.9	59.3	54.1	52.4	55.0	55.9	57.1		
		G-Mean (%)	50.2	54.9	47.3	54.1	51.7	55.0	48.5	52.2		
		Pfa (%)	33.1	43.3	76.5	31.0	39.4	43.4	26.5	65.2		
		CCD	DL	Precision (%)	90.9	93.3	0.0	74.3	87.1	91.6	63.6	84.1
				Recall (%)	73.5	86.4	0.0	77.3	87.1	91.6	49.4	61.6
F1-score (%)				81.3	89.7	0.0	75.8	87.1	91.6	55.6	71.1	
AUC (%)	84.9			92.0	0.0	86.5	91.4	94.1	70.0	78.6		
G-Mean (%)	84.2			91.8	0.0	91.9	91.3	94.1	66.9	76.6		
Pfa (%)	3.7			2.5	0.0	2.8	4.3	3.3	9.4	4.4		
CostEffect@5% (%)	54.5			50.0	80.0	52.0	45.2	48.4	58.9	61.5		
Popt@5% (%)	6.0			7.5	5.0	6.3	17.5	18.8	13.1	17.0		
CostEffect@20% (%)	10.8			13.0	9.4	11.2	25.2	27.0	21.5	26.6		
ML	Precision (%)		52.2	52.7	52.1	51.3	52.7	53.7	52.7	54.0		
	Recall (%)		24.3	27.1	22.3	24.5	39.2	40.8	34.8	39.3		
	F1-score (%)		1.6	1.3	7.1	3.8	12.1	11.4	7.8	9.0		
	AUC (%)		50.0	33.3	38.1	30.6	31.1	50.0	28.2	29.3		
	G-Mean (%)		28.0	30.8	16.0	82.7	37.1	42.1	21.0	40.0		
	Pfa (%)		35.9	32.0	22.5	44.6	33.8	45.7	24.1	33.7		
	CostEffect@5% (%)		57.0	56.7	54.3	55.5	54.9	54.7	51.7	50.6		
	Popt@5% (%)		49.1	50.4	38.5	48.4	51.9	53.2	41.6	49.5		
	CostEffect@20% (%)		14.0	17.3	7.4	71.7	27.1	22.6	17.6	38.6		
DP	DL	Precision (%)	0.0	81.8	0.0	0.0	74.0	88.3	54.9	57.0		
		Recall (%)	0.0	11.5	0.0	0.0	38.3	55.3	36.2	47.3		
		F1-score (%)	0.0	20.2	0.0	0.0	50.5	68.0	43.6	51.7		
		AUC (%)	0.0	55.3	0.0	0.0	63.8	74.7	56.2	59.4		
		G-Mean (%)	0.0	33.8	0.0	0.0	58.4	72.1	52.5	58.2		
		Pfa (%)	0.0	0.9	0.0	0.0	10.7	5.9	23.8	28.5		
		CostEffect@5% (%)	0.0	5.3	0.0	0.0	43.5	48.6	39.7	47.4		
		Popt@5% (%)	0.0	7.3	0.0	0.0	41.1	49.2	43.1	55.9		
		CostEffect@20% (%)	0.0	12.5	0.0	0.0	42.4	51.8	33.8	50.3		
	ML	Precision (%)	0.0	13.3	0.0	0.0	43.7	51.2	34.6	51.5		
		Recall (%)	50.0	33.3	38.1	30.6	31.1	50.0	28.2	29.3		
		F1-score (%)	28.0	30.8	16.0	82.7	37.1	42.1	21.0	40.0		
		AUC (%)	35.9	32.0	22.5	44.6	33.8	45.7	24.1	33.7		
		G-Mean (%)	57.0	56.7	54.3	55.5	54.9	54.7	51.7	50.6		
		Pfa (%)	49.1	50.4	38.5	48.4	51.9	53.2	41.6	49.5		
		CostEffect@5% (%)	14.0	17.3	7.4	71.7	27.1	22.6	17.6	38.6		
		CostEffect@20% (%)	0.0	13.3	0.0	0.0	43.7	51.2	34.6	51.5		
		Popt@20% (%)	0.0	13.3	0.0	0.0	43.7	51.2	34.6	51.5		
CCD	DL	Precision (%)	94.7	95.7	0.0	85.2	83.7	86.6	67.9	58.8		
		Recall (%)	61.0	70.0	0.0	61.0	83.1	92.1	27.5	84.2		
		F1-score (%)	74.2	80.7	0.0	72.2	84.9	89.2	36.9	69.3		
		AUC (%)	80.0	84.5	0.0	79.5	89.8	94.3	67.2	84.8		
		G-Mean (%)	77.8	83.2	0.0	77.3	89.5	94.2	51.1	84.8		
		Pfa (%)	0.8	0.8	0.0	2.0	3.4	6.6	13.1	4.4		
		CostEffect@5% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		Popt@5% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		CostEffect@20% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
	ML	Precision (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		Recall (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		F1-score (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		AUC (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		G-Mean (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		Pfa (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		CostEffect@5% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		Popt@5% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		CostEffect@20% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
DP	DL	Precision (%)	0.0	0.0	0.0	0.0	86.4	92.8	59.4	67.9		
		Recall (%)	0.0	0.0	0.0	0.0	40.0	62.0	41.4	51.8		
		F1-score (%)	0.0	0.0	0.0	0.0	54.8	74.3	48.8	58.8		
		AUC (%)	0.0	0.0	0.0	0.0	67.5	79.1	59.4	66.1		
		G-Mean (%)	0.0	0.0	0.0	0.0	61.7	77.2	56.6	64.5		
		Pfa (%)	0.0	0.0	0.0	0.0	5.1	3.8	19.6	22.7		
		CostEffect@5% (%)	0.0	0.0	0.0	0.0	32.0	40.9	35.1	38.3		
		Popt@5% (%)	0.0	0.0	0.0	0.0	33.8	45.1	36.2	41.6		
		CostEffect@20% (%)	0.0	0.0	0.0	0.0	33.8	40.4	33.5	38.7		
	ML	Precision (%)	36.7	33.3	35.3	66.7	37.9	31.7	32.1	25.0		
		Recall (%)	88.3	68.8	97.7	16.7	86.0	63.8	81.8	30.2		
		F1-score (%)	51.9	28.6	51.9	26.7	52.7	32.4	46.2	27.4		
		AUC (%)	56.7	55.1	54.8	57.3	58.5	53.2	49.6	54.0		
		G-Mean (%)	47.1	50.6	34.2	40.4	51.5	52.1	37.7	48.5		
		Pfa (%)	74.9	23.1	88.0	2.0	69.1	57.4	49.6	22.3		
		CostEffect@5% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		Popt@5% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
		CostEffect@20% (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
CCD	DL	Cliff's d	0.56	—	1.0	0.56	0.56	—	1.0	0.89		
	ML	Cliff's d	0.22	—	0.22	0.22	0.22	—	0.44	0.0		
DP	DL	Cliff's d	0.33	—	0.56	0.33	1.0	—	1.0	1.0		
	ML	Cliff's d	1.0	—	1.0	0.56	0.56	—	1.0	0.78		

TABLE V  
THE PERFORMANCE OF VARIOUS TRAINING METHODS ON DATASETS WITH LOW SCALE AND DIVERSE DATA BALANCE DISTRIBUTIONS

Task	Model	Distribution Method	LH							
		VFL	ALMITY	CTM <sub>client</sub>	CTM <sub>all</sub>	VFL+Oversampling	ALMITY+Oversampling	CTM <sub>client</sub> +Oversampling	CTM <sub>all</sub> +Oversampling	
CCD	DL	Precision (%)	87.8	89.1	0.0	88.9	80.4	83.3	55.1	71.1
		Recall (%)	80.0	91.1	0.0	87.9	88.2	93.2	27.1	65.2
		F1-score (%)	83.7	90.1	0.0	88.4	84.1	88.0	36.4	68.0
		AUC (%)	86.3	91.9	0.0	91.8	91.4	94.3	60.8	72.1
		G-Mean (%)	86.1	91.9	0.0	91.7	91.4	94.3	50.6	71.8
		Pfa (%)	7.4	7.4	0.0	9.1	5.4	4.6	5.5	20.9
	ML	Precision (%)	79.3	57.1	53.8	67.9	55.8	51.0	50.0	51.0
		Recall (%)	9.0	10.0	4.4	9.1	51.6	55.6	24.0	65.2
		F1-score (%)	16.1	17.0	8.1	16.1	53.6	57.6	32.4	57.3
		AUC (%)	58.5	64.5	50.3	58.0	58.4	59.0	57.3	56.7
		G-Mean (%)	29.7	30.9	20.5	29.7	57.1	57.6	43.8	57.1
		Pfa (%)	1.9	4.3	3.8	3.4	35.6	37.5	20.0	50.0
DP	DL	Precision (%)	53.2	50.0	0.0	71.4	53.5	62.9	50.3	50.5
		Recall (%)	20.0	34.6	0.0	1.6	43.1	54.8	23.8	36.0
		F1-score (%)	29.1	40.9	0.0	3.2	50.9	58.6	32.3	42.0
		AUC (%)	51.2	52.8	0.0	50.6	54.5	54.5	52.1	53.9
		G-Mean (%)	40.6	49.6	0.0	12.7	50.9	57.7	43.7	50.8
		Pfa (%)	17.6	29.0	0.0	0.5	24.0	20.8	19.6	28.3
	ML	CostEffect@5% (%)	25.0	28.1	0.0	0.0	36.1	63.6	17.8	44.4
		Popt@5% (%)	27.2	30.4	0.0	0.0	39.2	66.6	17.5	39.2
		CostEffect@20% (%)	18.6	25.7	0.0	2.4	31.6	56.2	19.2	29.8
		Popt@20% (%)	20.3	26.1	0.0	2.6	33.0	57.5	18.7	30.7
		Precision (%)	42.3	42.2	46.1	41.7	55.7	55.1	55.8	56.6
		Recall (%)	61.1	66.0	89.4	38.0	89.6	76.0	98.9	94.3
CCD	DL	Precision (%)	84.6	86.8	0.0	0.0	65.5	83.9	50.0	71.6
		Recall (%)	50.0	65.0	0.0	0.0	86.4	76.5	17.6	58.5
		F1-score (%)	62.9	74.5	0.0	0.0	62.9	80.0	26.1	64.4
		AUC (%)	73.5	77.0	0.0	0.0	85.8	86.1	54.4	74.8
		G-Mean (%)	69.7	72.2	0.0	0.0	85.8	86.0	40.1	73.0
		Pfa (%)	2.9	1.0	0.0	0.0	14.7	4.2	8.8	8.9
	ML	Precision (%)	0.0	0.0	0.0	0.0	63.6	62.5	50.0	66.7
		Recall (%)	0.0	0.0	0.0	0.0	3.8	5.0	2.9	4.2
		F1-score (%)	0.0	0.0	0.0	0.0	7.2	9.3	5.4	7.8
		AUC (%)	0.0	0.0	0.0	0.0	57.3	59.2	66.7	56.5
		G-Mean (%)	0.0	0.0	0.0	0.0	19.3	22.1	16.8	20.2
		Pfa (%)	0.0	0.0	0.0	0.0	1.7	2.1	1.4	1.7
DP	DL	Precision (%)	0.0	100.0	0.0	0.0	64.8	69.5	54.4	53.3
		Recall (%)	0.0	3.2	0.0	0.0	32.0	37.6	23.7	20.9
		F1-score (%)	0.0	6.3	0.0	0.0	42.8	49.6	33.0	30.0
		AUC (%)	0.0	51.6	0.0	0.0	59.1	58.9	56.9	53.1
		G-Mean (%)	0.0	18.0	0.0	0.0	52.5	58.5	44.7	42.2
		Pfa (%)	0.0	0.2	0.0	0.0	13.9	9.8	15.9	14.7
	ML	CostEffect@5% (%)	0.0	0.0	0.0	0.0	44.5	52.9	27.0	34.3
		Popt@5% (%)	0.0	0.0	0.0	0.0	42.8	49.9	28.6	32.3
		CostEffect@20% (%)	0.0	12.5	0.0	0.0	45.4	53.4	22.3	40.0
		Popt@20% (%)	0.0	12.3	0.0	0.0	45.3	52.3	22.1	32.4
		Precision (%)	47.7	45.5	40.5	28.7	23.8	31.5	35.7	28.6
		Recall (%)	78.0	89.1	87.5	70.0	83.3	85.7	83.3	33.3
CCD	DL	Precision (%)	0.0	0.0	0.0	0.0	59.4	74.1	0.0	36.0
		Recall (%)	0.0	0.0	0.0	0.0	76.0	80.0	0.0	68.5
		F1-score (%)	0.0	0.0	0.0	0.0	66.7	76.9	0.0	47.2
		AUC (%)	0.0	0.0	0.0	0.0	81.6	86.6	0.0	69.2
		G-Mean (%)	0.0	0.0	0.0	0.0	81.4	86.3	0.0	69.2
		Pfa (%)	0.0	0.0	0.0	0.0	12.6	6.9	0.0	10.1
	ML	Precision (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		Recall (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		F1-score (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		AUC (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		G-Mean (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		Pfa (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DP	DL	Precision (%)	0.0	0.0	0.0	0.0	75.4	88.3	45.8	47.2
		Recall (%)	0.0	0.0	0.0	0.0	32.3	37.5	34.8	32.5
		F1-score (%)	0.0	0.0	0.0	0.0	45.2	52.6	39.6	38.1
		AUC (%)	0.0	0.0	0.0	0.0	51.9	66.8	51.0	53.8
		G-Mean (%)	0.0	0.0	0.0	0.0	54.4	60.0	38.3	40.5
		Pfa (%)	0.0	0.0	0.0	0.0	8.4	4.0	32.9	6.9
	ML	CostEffect@5% (%)	0.0	0.0	0.0	0.0	47.6	65.5	37.6	25.0
		Popt@5% (%)	0.0	0.0	0.0	0.0	50.5	67.5	41.7	28.7
		CostEffect@20% (%)	0.0	0.0	0.0	0.0	44.3	57.9	38.7	12.5
		Popt@20% (%)	0.0	0.0	0.0	0.0	46.0	59.1	40.0	26.7
		Precision (%)	20.0	40.5	25.0	46.2	32.7	34.0	33.2	36.3
		Recall (%)	96.7	95.8	9.5	75.0	76.2	81.0	84.2	19.0
CCD	DL	Precision (%)	0.33	—	0.67	0.56	1.0	—	1.0	1.0
		Recall (%)	0.33	—	0.56	0.56	0.56	—	0.78	0.56
		F1-score (%)	—	—	—	—	—	—	—	—
		AUC (%)	—	—	—	—	—	—	—	—
		G-Mean (%)	—	—	—	—	—	—	—	—
		Pfa (%)	—	—	—	—	—	—	—	—
	ML	Cliff's d	0.50	—	1.0	0.5	0.56	—	1.0	1.0
		Cliff's d	0.11	—	0.11	0.11	0.11	—	0.22	0.11
		Cliff's d	—	—	—	—	—	—	—	—
		Cliff's d	—	—	—	—	—	—	—	—
		Cliff's d	—	—	—	—	—	—	—	—
		Cliff's d	—	—	—	—	—	—	—	—

TABLE VI  
THE PERFORMANCE OF VARIOUS TRAINING METHODS ON DATASETS WITH DIVERSE DATA DISTRIBUTIONS

Task	Model	#Clients&Distribution Method	9 & HH, HM, HL, MH, MM, ML, LH, LM, LL					
			VFL	ALMITY	CTM <sub>all</sub>	VFL+Oversampling	ALMITY+Oversampling	CTM <sub>all</sub> +Oversampling
CCD	DL	Precision (%)	87.7	89.4	91.3	80.9	96.7	72.4
		Recall (%)	73.5	86.8	75.9	90.4	91.9	80.6
		F1-score (%)	80.0	88.1	82.8	85.4	94.7	76.3
		AUC (%)	84.2	90.8	88.8	87.9	94.4	81.9
		G-Mean (%)	83.5	90.7	87.8	87.9	94.3	81.9
		Pfa (%)	5.1	5.1	5.3	10.7	1.1	16.8
	ML	Precision (%)	22.2	83.3	66.6	33.3	27.3	57.4
		Recall (%)	5.9	6.7	5.7	11.8	17.6	13.0
		F1-score (%)	9.3	12.4	10.5	17.4	21.4	21.1
		AUC (%)	47.9	52.8	52.1	50.2	47.4	52.4
		G-Mean (%)	23.0	25.7	23.7	32.3	36.9	34.5
		Pfa (%)	10.0	1.1	1.4	11.4	12.8	8.1
DP	DL	Precision (%)	66.7	55.9	53.8	69.0	71.6	56.4
		Recall (%)	14.5	15.2	12.5	45.9	53.0	41.7
		F1-score (%)	21.8	23.9	21.6	55.1	64.2	47.9
		AUC (%)	52.4	54.6	53.7	64.7	67.7	57.9
		G-Mean (%)	35.2	37.8	35.6	61.9	64.2	55.6
		Pfa (%)	7.7	6.0	11.2	16.5	9.6	26.1
		CostEffect@5% (%)	9.7	17.9	13.6	38.7	50.7	38.5
		Popt@5% (%)	9.0	17.7	14.6	40.4	51.4	39.5
	ML	CostEffect@20% (%)	2.2	13.9	16.4	40.0	52.6	40.4
		Popt@20% (%)	5.0	16.3	17.0	40.2	52.7	41.2
		Precision (%)	48.9	49.0	33.3	37.7	44.1	33.8
		Recall (%)	26.5	38.3	27.3	90.9	49.4	86.5
ML	F1-score (%)	34.4	42.9	30.0	53.3	46.6	52.6	
	AUC (%)	51.6	53.0	56.8	58.0	49.4	57.2	
	G-Mean (%)	45.1	50.9	48.5	47.6	49.4	47.5	
	Pfa (%)	23.2	32.3	13.6	75.0	50.6	55.4	
CCD	DL	Cliff's d	1.0	—	1.0	1.0	—	1.0
	ML	Cliff's d	1.0	—	1.0	1.0	—	1.0
DP	DL	Cliff's d	1.0	—	1.0	1.0	—	1.0
	ML	Cliff's d	1.0	—	1.0	1.0	—	1.0

Even in cases where ALMITY cannot achieve satisfactory performance, such as a G-Mean of 0.0, it still surpasses the baselines' performance after the application of the oversampling technique.

- 6) **Experiment 1: The oversampling technique yields performance enhancements of varying degrees for training methods within diverse data distributions.** Examining Tables III to V, we can discern that in relatively balanced datasets like the HH data distribution, the oversampling technique yields only marginal performance improvements in learning-based models, particularly deep learning (DL) models, across two evaluation tasks. Conversely, in significantly skewed data distributions, the oversampling technique elevates model performance. For example, consider the HH data distribution where ML models demonstrate only marginal performance enhancements, typically in the range of 3-4%, when incorporating the oversampling technique in the CCD task. The DL-based clone detector experiences a slight performance decline after its application. One possible explanation for this decline in performance is that the oversampling technique leads to an abundance of duplicated samples in the dataset, causing overfitting of the DL model during training. Conversely, within the LH data distribution, the oversampling technique leads

to substantial improvements. Specifically, it results in a remarkable 20% boost in performance for the ML-based clone detector in terms of G-Mean, while the oversampling technique makes approximately 10% and over 5% enhancements for DL-based and ML-based defect predictors, respectively. Note that the strong performance of the DL-based clone detector in skewed data distributions can be attributed to the CCD task's inherent characteristics. Even within datasets exhibiting substantial skew, a enough number of data instances exist for each class, enabling the DL model to proficiently acquire the capability to distinguish between clones and non-clones. This underscores the notion that identifying similar code fragments is not a challenging task for DL models.

- 7) **Experiment 2: Despite the involvement of multiple datasets with diverse data distributions in training learning-based models, ALMITY remains effective in enhancing the overall performance of these models.** In real-world applications, client-side datasets within the FL-based framework frequently encompass diverse data distributions, encompassing variations in both data scale and data balance attributes. To illustrate the utility and effectiveness of ALMITY in such scenarios, we conduct an evaluation of various training methods on datasets involving all these data distributions concurrently. ALMITY

consistently empowers both ML and DL models to outperform the baselines, as evidenced by the uniformly high Cliff's  $d$  values observed in both the CCD and DP tasks.

- 8) **Experiments 1 and 2: In most instances, ALMITY outperforms VFL and CTM training methods in terms of effort-aware metrics.** From these four tables, we can observe that ALMITY achieves higher CostEffort and Popt values than baselines. This is attributed to the better performance of ALMITY in AUC and G-Mean. As more defects are identified from datasets by ALMITY, the effort-aware metrics naturally increase. While  $CTM_{all}$  may occasionally yield higher effort-aware metric values than ALMITY, it is important to note that  $CTM_{all}$  is not a realistic scenario. In practice, different clients in ALMITY often represent distinct organizations or companies, and integrating their proprietary industrial datasets is unfeasible.
- 9) **Statistical test analysis: ALMITY's performance in training learning-based models exhibits statistical significance when compared to the baseline methods in both CCD and DP tasks.** Table VII reveals that while ALMITY's performance may not much exceed that of the baselines in certain distributions, all  $p$ -values resulting from pairwise comparisons between ALMITY and each baseline method are below 0.05. The highest  $p$ -value obtained from the pairwise analysis between ALMITY and VFL's performance on the DL-based defect predictor is  $4.14 \times 10^{-2}$ . This shows that the null hypothesis that ALMITY cannot achieve better performance than baselines can be rejected at a confidence level of 5% in favor of the alternative that ALMITY would tend to achieve better performance than baseline training methods. Therefore, we can conclude that ALMITY's performance in training learning-based models is statistically superior to that of the baseline methods for both CCD and DP tasks.

🔗 **RQ-2** ▶ *We have undertaken comprehensive experiments to assess the effectiveness of employing ALMITY for training DL and ML models across a spectrum of diverse data distributions. The experimental outcomes reveal that ALMITY surpasses the performance of baseline methods in both evaluation tasks. ALMITY enhances model performance across nearly all data distributions, thereby substantiating the effectiveness of our parameter aggregation strategy. Furthermore, to substantiate ALMITY's proficiency in handling complex real-world datasets, we conduct experiments validating its effectiveness across task-specific data distributions and on multiple datasets featuring a variety of data distribution types.* ◀

## VII. RQ-3: HOW DO INDIVIDUAL ATTRIBUTES AFFECT ALMITY'S EFFECTIVENESS?

**Motivation.** Our parameter aggregation strategy has been meticulously crafted by strategically integrating three distinct attributes. As a result, in this RQ, we undertake ablation

experiments to scrutinize the individual impact of each attribute on ALMITY's performance.

**Method.** To accomplish the objectives of this RQ, we have conducted three rounds of experiments. In each experimental round, we systematically exclude one attribute from our parameter aggregation strategy to evaluate its specific impact on the performance of ALMITY. For example, given that our aggregation strategy relies on the interaction of three parameters—namely, data scale, data balance, and minority class learnability attributes—we proceed to investigate the impact of the data scale attribute by temporarily removing it and thus modifying our strategy into the integration of the remaining two attributes. Hence, a low performance indicates the high significance of a particular attribute within our aggregation strategy.

We conduct these three rounds of experiments across six distinct data distributions. Specifically, we employ the HH and MM data distributions to gauge the influence of each attribute on datasets with comparable data size and balance. The HM and HL data distributions are chosen to assess attribute performance in datasets characterized by lower data balance, while the MH and LH data distributions allow us to evaluate each attribute's impact on datasets with diminished data scale. Additionally, among these six data distributions, two pairs exhibit opposing characteristics: the HM and MH data distributions, as well as the HL and LH data distributions. Note that the LM, ML, and LL data distributions are excluded from our experiments due to the absence of minority class instances or all-class data instances in these distributions, making it impractical to measure the impact of each attribute.

**Metrics.** In this RQ, we utilize a consistent set of evaluation metrics, as employed in the preceding two RQs. These metrics include Precision, Recall, F1-score, AUC, G-Mean, and the Probability of False Alarm (Pfa) to gauge the model's performance. Furthermore, we extend our analysis to encompass cost-effectiveness, as measured by the metrics CostEffect and Popt at both 5% and 20%, specifically within the context of code churn.

**Results.** Drawing conclusions from the experimental results presented in Table VIII, we can summarize as follows:

- 1) **Together, these three indicators contribute positively to our parameter aggregation strategy.** Referencing Table VIII, across all types of data distributions, it is evident that the performance of ALMITY on both ML and DL models consistently diminishes regardless of which attribute is excluded from the aggregation strategy. As an illustration, considering the HH data distribution, the performance of ALMITY exhibits a reduction of at least 4% and 2% on ML models, and an average decrease of approximately 2% and 3% on DL models in these two evaluation tasks, respectively. Furthermore, in the HL data distribution, when any attribute is omitted from the parameter strategy, ALMITY fails to correctly identify defects and clones from the corresponding datasets using DL models, resulting in a G-Mean of 0.0. These observations confirm the pivotal role played by all three attributes within our parameter strategy.

TABLE VII  
WILCOXON SIGNED-RANK TEST PAIRWISE ANALYSIS ON THE PERFORMANCE OF DIFFERENT TRAINING METHODS ACROSS DIVERSE DATA DISTRIBUTIONS

Task	Model	Methods	$p$ -value	Methods	$p$ -value
CCD	DL	ALMITY & VFL	$0.45 \times 10^{-2}$	ALMITY + oversampling & VFL + oversampling	$0.19 \times 10^{-2}$
		ALMITY & CTM <sub>single</sub>	$0.45 \times 10^{-2}$	ALMITY + oversampling & CTM <sub>single</sub> + oversampling	$0.19 \times 10^{-2}$
		ALMITY & CTM <sub>all</sub>	$0.89 \times 10^{-2}$	ALMITY + oversampling & CTM <sub>all</sub> + oversampling	$0.58 \times 10^{-2}$
	ML	ALMITY & VFL	$1.73 \times 10^{-2}$	ALMITY + oversampling & VFL + oversampling	$1.02 \times 10^{-2}$
		ALMITY & CTM <sub>single</sub>	$1.73 \times 10^{-2}$	ALMITY + oversampling & CTM <sub>single</sub> + oversampling	$1.02 \times 10^{-2}$
		ALMITY & CTM <sub>all</sub>	$1.73 \times 10^{-2}$	ALMITY + oversampling & CTM <sub>all</sub> + oversampling	$3.26 \times 10^{-2}$
DP	DL	ALMITY & VFL	$4.14 \times 10^{-2}$	ALMITY + oversampling & VFL + oversampling	$0.19 \times 10^{-2}$
		ALMITY & CTM <sub>single</sub>	$0.40 \times 10^{-2}$	ALMITY + oversampling & CTM <sub>single</sub> + oversampling	$0.19 \times 10^{-2}$
		ALMITY & CTM <sub>all</sub>	$0.40 \times 10^{-2}$	ALMITY + oversampling & CTM <sub>all</sub> + oversampling	$0.19 \times 10^{-2}$
	ML	ALMITY & VFL	$0.19 \times 10^{-2}$	ALMITY + oversampling & VFL + oversampling	$0.19 \times 10^{-2}$
		ALMITY & CTM <sub>single</sub>	$0.19 \times 10^{-2}$	ALMITY + oversampling & CTM <sub>single</sub> + oversampling	$0.19 \times 10^{-2}$
		ALMITY & CTM <sub>all</sub>	$0.19 \times 10^{-2}$	ALMITY + oversampling & CTM <sub>all</sub> + oversampling	$0.19 \times 10^{-2}$

- 2) **Of these three attributes, the data scale attribute exerts the most significant impact on the parameter aggregation strategy.** According to Table VIII, in both the CCD and DP tasks, ALMITY's performance experiences a decline when dealing with data distributions characterized by low data scales. For instance, consider the MH data distribution, where the data scale is slightly lower than the data balance value. Upon removing the influence of the data scale attribute, ALMITY's performance in this distribution registers its lowest point, with a G-Mean decrease of over 11% in the DP task. This emphasizes the substantial impact of the data scale attribute within the aggregation strategy. Besides, in data distributions with comparable data scale and data balance values, such as the HH and MM distributions, excluding the data scale attribute still leads to a substantial performance drop for ALMITY. For example, in the MM data distribution, ALMITY's performance without the data scale attribute reaches only 0.0% and 40.7% in terms of G-Mean for the CCD and DP tasks, respectively.
- 3) **Generally, when an attribute exhibits lower values within the data distribution, it tends to have a more pronounced impact on ALMITY's performance.** As elucidated in the previous discovery, in data distributions characterized by low data scales, the removal of the data scale attribute has the most significant impact on ALMITY's overall performance. Similarly, in data distributions with low data balance, the data balance attribute demonstrates an important influence on ALMITY's performance. For instance, in the DP task, within the HM data distribution, the exclusion of the data balance attribute results in an 11.1% decline in ALMITY's performance on the DL model. These findings collectively underscore that when an attribute exhibits lower values within a data distribution, this attribute tends to exert a more noticeable impact on ALMITY's performance.
- 4) **The minority class learnability attribute is influenced by both the data scale and data balance values within the data distribution. When dealing with data distributions that have comparable data scales**

**and data balances, the minority class learnability attribute primarily affects ALMITY's performance.** It is readily comprehensible that the minority class learnability attribute is influenced by both the data scale and data balance values within a given data distribution. This is attributed to the fluctuations in the number of minority class data instances within a dataset, which occur as a consequence of changes in data scale and data balance within the dataset. Drawing insights from the experimental results presented in Table VIII, in the CCD and DP tasks, we observe that within the HH and MM data distributions, characterized by comparable data scale and data balance, ALMITY exhibits its poorest performance on both ML and DL models when the minority class learnability attribute is removed.

✎ **RQ-3** ▶ *In this RQ, we have observed that each attribute contributes positively to ALMITY's performance, highlighting the essential role of these three attributes within our aggregation strategy. Furthermore, we note that each attribute exhibits a more pronounced impact on specific data distribution types, with the data scale attribute having a relatively significant influence among the three.* ◀

## VIII. DISCUSSION

In this section, based on the insights gained from ALMITY, we explore extensive application scenarios of our approach. In addition, we delineate the usage limitations of ALMITY.

**Application Scenarios.** We explore the potential real-world applications of ALMITY. It stands out as a task-agnostic and model-agnostic training framework, characterized by the following key attributes:

- 1) **Binary Classification-Specific:** ALMITY is tailored for binary classification tasks. It can be effectively employed in various tasks, such as code classification, vulnerability detection, etc.
- 2) **Model-Agnostic:** ALMITY can be integrated into various learning-based models, encompassing both ML and DL models. Nevertheless, it is important to highlight that, like other FL-based frameworks, ALMITY requires

TABLE VIII  
ABLATION EXPERIMENTS: THE IMPACT OF EACH ATTRIBUTE ON THE PERFORMANCE OF ALMITY

Model	Task	Distribution Method	HH				Model	Task	Distribution Method	HH			
			ALMITY	-Data Scale	-Data Balance	-Data Learnability				ALMITY	-Data Scale	-Data Balance	-Data Learnability
DL	CCD	Precision (%)	96.3	95.5	95.6	95.5	ML	CCD	Precision (%)	49.2	47.8	53.5	53.3
		Recall (%)	93.8	91.3	91.6	91.3			Recall (%)	51.8	67.5	37.3	33.3
		F1-score (%)	95.0	93.4	93.6	93.3			F1-score (%)	50.4	56.0	43.9	41.0
		AUC (%)	95.6	93.9	93.9	93.8			AUC (%)	56.8	57.5	57.1	55.0
		G-Mean (%)	95.6	93.9	93.9	93.8			G-Mean (%)	56.6	55.8	53.5	50.6
		Pfa (%)	2.5	3.6	3.8	3.6			Pfa (%)	38.1	52.5	23.1	23.3
	DP	Precision (%)	59.0	58.6	64.2	54.1	ML	DP	Precision (%)	59.9	40.0	44.2	44.6
		Recall (%)	51.7	43.4	43.7	43.0			Recall (%)	56.5	69.2	47.5	78.4
		F1-score (%)	55.1	49.9	52.0	47.9			F1-score (%)	52.2	50.7	45.8	56.9
		AUC (%)	63.0	60.0	60.8	57.5			AUC (%)	57.0	58.7	53.2	54.6
		G-Mean (%)	62.0	57.6	58.3	55.7			G-Mean (%)	57.0	53.6	54.9	51.1
		Pfa (%)	25.6	23.6	22.0	27.9			Pfa (%)	45.8	57.6	41.1	69.2
CostEffect@5%	37.5	36.0	36.8	32.5	CostEffect@5%	—	—	—	—				
Popt@5%	42.5	41.3	42.0	36.4	Popt@5%	—	—	—	—				
CostEffect@20%	39.1	37.0	39.0	33.6	CostEffect@20%	—	—	—	—				
Popt@20%	37.9	36.4	37.4	32.9	Popt@20%	—	—	—	—				
DL	CCD	Precision (%)	93.3	87.5	90.9	94.4	ML	CCD	Precision (%)	50.0	0.0	100.0	0.0
		Recall (%)	86.4	86.4	86.4	82.7			Recall (%)	7.5	0.0	3.8	0.0
		F1-score (%)	89.7	87.0	88.6	88.2			F1-score (%)	13.0	0.0	7.2	0.0
		AUC (%)	92.0	90.8	91.5	90.4			AUC (%)	52.7	0.0	51.9	0.0
		G-Mean (%)	91.9	90.7	91.4	90.0			G-Mean (%)	27.1	0.0	19.4	0.0
		Pfa (%)	2.5	4.9	3.4	2.0			Pfa (%)	1.3	0.0	0.0	0.0
	DP	Precision (%)	81.8	0.0	33.3	40.0	ML	DP	Precision (%)	33.3	20.5	20.9	20.9
		Recall (%)	11.5	0.0	1.6	1.6			Recall (%)	30.8	69.2	69.2	69.2
		F1-score (%)	20.2	0.0	3.1	3.1			F1-score (%)	32.0	31.6	32.1	31.6
		AUC (%)	55.3	0.0	50.0	50.2			AUC (%)	56.7	46.6	47.7	46.6
		G-Mean (%)	33.8	0.0	12.5	12.6			G-Mean (%)	50.4	40.7	42.5	40.7
		Pfa (%)	0.8	0.0	1.6	1.2			Pfa (%)	17.3	76.1	73.9	76.1
CostEffect@5%	5.3	0.0	0.0	0.0	CostEffect@5%	—	—	—	—				
Popt@5%	7.3	0.0	0.0	0.0	Popt@5%	—	—	—	—				
CostEffect@20%	12.5	0.0	8.5	9.3	CostEffect@20%	—	—	—	—				
Popt@20%	13.3	0.0	8.5	9.3	Popt@20%	—	—	—	—				
DL	CCD	Precision (%)	92.1	88.9	91.4	86.5	ML	CCD	Precision (%)	44.6	49.5	37.9	70.9
		Recall (%)	86.9	85.0	80.4	85.0			Recall (%)	38.0	27.5	84.6	17.2
		F1-score (%)	89.4	86.9	85.5	85.7			F1-score (%)	41.0	35.4	52.4	27.7
		AUC (%)	92.0	90.7	89.1	90.2			AUC (%)	49.0	53.8	53.2	57.2
		G-Mean (%)	91.8	90.5	88.7	90.1			G-Mean (%)	47.8	46.9	42.8	40.8
		Pfa (%)	2.9	3.5	2.2	4.4			Pfa (%)	39.9	20.0	78.3	2.8
	DP	Precision (%)	64.7	55.6	53.8	63.6	ML	DP	Precision (%)	34.1	48.3	46.8	32.1
		Recall (%)	10.6	6.0	5.0	5.0			Recall (%)	77.5	80.6	27.5	78.7
		F1-score (%)	18.2	10.8	9.2	9.3			F1-score (%)	47.3	60.4	34.6	45.7
		AUC (%)	54.3	52.3	51.4	51.8			AUC (%)	58.7	56.6	57.5	56.1
		G-Mean (%)	32.2	24.2	22.1	22.2			G-Mean (%)	53.7	52.2	49.1	51.3
		Pfa (%)	1.9	1.3	2.1	1.4			Pfa (%)	60.0	67.4	12.5	66.5
CostEffect@5%	11.9	11.9	11.9	13.9	CostEffect@5%	—	—	—	—				
Popt@5%	8.1	8.0	8.0	8.0	Popt@5%	—	—	—	—				
CostEffect@20%	9.1	7.9	5.5	4.8	CostEffect@20%	—	—	—	—				
Popt@20%	8.9	7.7	5.5	4.8	Popt@20%	—	—	—	—				
DL	CCD	Precision (%)	93.7	87.9	91.8	95.3	ML	CCD	Precision (%)	56.9	69.0	22.2	56.5
		Recall (%)	90.6	87.9	90.1	87.3			Recall (%)	33.0	12.9	62.5	26.0
		F1-score (%)	92.1	87.9	91.0	91.0			F1-score (%)	41.8	21.8	32.8	35.6
		AUC (%)	92.8	89.3	91.7	91.9			AUC (%)	56.5	54.2	50.0	55.9
		G-Mean (%)	92.7	89.3	91.7	91.8			G-Mean (%)	51.4	35.1	48.4	47.2
		Pfa (%)	5.1	9.3	6.7	3.4			Pfa (%)	11.4	4.6	62.5	14.3
	DP	Precision (%)	66.7	55.8	56.6	54.3	ML	DP	Precision (%)	48.2	48.4	44.9	44.2
		Recall (%)	46.2	33.3	42.1	35.4			Recall (%)	45.0	86.1	69.2	47.5
		F1-score (%)	54.5	41.7	48.2	42.9			F1-score (%)	46.6	62.0	54.4	45.8
		AUC (%)	64.9	56.5	57.6	56.2			AUC (%)	55.9	57.2	55.4	53.2
		G-Mean (%)	62.1	51.5	55.4	52.2			G-Mean (%)	54.9	49.3	54.4	52.9
		Pfa (%)	16.4	20.3	26.9	23.0			Pfa (%)	47.3	71.7	58.3	41.1
CostEffect@5%	54.3	35.1	54.1	53.6	CostEffect@5%	—	—	—	—				
Popt@5%	52.9	37.2	51.3	51.6	Popt@5%	—	—	—	—				
CostEffect@20%	45.8	38.2	45.0	42.8	CostEffect@20%	—	—	—	—				
Popt@20%	47.0	37.5	45.2	45.5	Popt@20%	—	—	—	—				
DL	CCD	Precision (%)	98.6	95.6	93.5	93.5	ML	CCD	Precision (%)	0.0	0.0	0.0	0.0
		Recall (%)	83.3	77.8	76.3	76.3			Recall (%)	0.0	0.0	0.0	0.0
		F1-score (%)	90.6	85.7	84.1	84.1			F1-score (%)	0.0	0.0	0.0	0.0
		AUC (%)	91.8	88.4	87.5	87.5			AUC (%)	0.0	0.0	0.0	0.0
		G-Mean (%)	91.4	87.7	86.8	86.8			G-Mean (%)	0.0	0.0	0.0	0.0
		Pfa (%)	0.3	8.7	13.1	13.1			Pfa (%)	0.0	0.0	0.0	0.0
	DP	Precision (%)	64.3	0.0	0.0	0.0	ML	DP	Precision (%)	16.9	30.8	0.0	60.0
		Recall (%)	3.2	0.0	0.0	0.0			Recall (%)	19.6	8.0	0.0	6.0
		F1-score (%)	6.4	0.0	0.0	0.0			F1-score (%)	18.2	12.7	0.0	10.9
		AUC (%)	50.6	0.0	0.0	0.0			AUC (%)	47.8	51.8	0.0	52.5
		G-Mean (%)	16.7	0.0	0.0	0.0			G-Mean (%)	38.6	27.6	0.0	24.4
		Pfa (%)	2.2	0.0	0.0	0.0			Pfa (%)	7.0	4.5	0.0	1.0
CostEffect@5%	6.3	0.0	0.0	0.0	CostEffect@5%	—	—	—	—				
Popt@5%	2.4	0.0	0.0	0.0	Popt@5%	—	—	—	—				
CostEffect@20%	2.3	0.0	0.0	0.0	CostEffect@20%	—	—	—	—				
Popt@20%	2.7	0.0	0.0	0.0	Popt@20%	—	—	—	—				
DL	CCD	Precision (%)	89.1	82.9	87.0	86.7	ML	CCD	Precision (%)	57.1	50.0	50.0	66.7
		Recall (%)	91.1	64.4	88.9	86.7			Recall (%)	10.0	1.2	9.8	4.9
		F1-score (%)	90.1	72.5	87.9	86.7			F1-score (%)	17.0	2.4	16.3	9.1
		AUC (%)	91.9	77.8	90.0	88.9			AUC (%)	64.5	50.3	52.0	51.7
		G-Mean (%)	91.9	76.7	90.0	88.9			G-Mean (%)	30.9	11.0	30.3	21.9
		Pfa (%)	7.4	8.8	8.8	8.8			Pfa (%)	4.3	0.7	5.7	1.4
	DP	Precision (%)	50.0	50.0	57.6	43.9	ML	DP	Precision (%)	42.2	41.6	37.9	42.3
		Recall (%)	34.6	26.8	30.4	28.1			Recall (%)	66.0	90.9	84.6	61.1
		F1-score (%)	40.9	34.9	39.8	34.3			F1-score (%)	58.7	57.1	52.4	50.0
		AUC (%)	52.8	54.5	54.0	53.5			AUC (%)	59.5	53.5	53.2	47.9
		G-Mean (%)	49.6	47.0	48.6	47.1			G-Mean (%)	47.0	38.1	42.9	46.1
		Pfa (%)	29.0	17.7	22.4	21.1			Pfa (%)	63.0	84.0	78.3	65.2
CostEffect@5%	28.1	25.0	34.3	28.1	CostEffect@5%	—	—	—	—				
Popt@5%	30.4	27.3	39.9	30.3	Popt@5%	—	—	—	—				
CostEffect@20%	25.7	23.2	29.6	24.1	CostEffect@20%	—	—	—	—				
Popt@20%	26.1	23.9	31.8	25.3	Popt@20%	—	—	—	—				

all clients to deploy identical models for proper functioning; deviations from this standard can disrupt the framework's operation.

- 3) **Sensitive-Agnostic:** ALMITY is suitable for deployment in sensitive fields with data scarcity. This is attributed to the fact that ALMITY operates without the need for sensitive information from datasets, such as detailed data instances. It enhances the performance of learning-based models with imbalanced datasets while minimizing the demand for extensive data information, only requiring the number of positives and negatives from datasets.
- 4) **Participant-Agnostic:** Our approach transcends the boundaries of participants, extending beyond researchers and industrial practitioners. ALMITY can be effectively employed by diverse organizations seeking collaborative global model training, including small enterprises, without restriction.

**What data should or should not be uploaded.** Note that in the practical implementation of ALMITY, a client is only required to upload the model parameters after each training round and the count of data instances for each class within the corresponding client-side dataset. There is no necessity to transmit specific data instances to the server. This is because the information regarding the number of data instances for each class within the dataset is sufficient for the server to calculate the data scale, data balance, and minority class learnability attributes. From this explanation, readers can readily discern that, in comparison to the VFL framework, which necessitates clients to transmit model parameters after each training round and the total count of data instances within the client-side datasets, ALMITY demands only a minimal increase in information transmission but is capable of achieving superior performance in training learning-based models.

**Usage Limitations of ALMITY.** Similar to other FL-based frameworks, our approach assumes that the collaborated clients share the same features, which causes the organizations having different feature spaces might fail to directly apply our approach in practice. In cases where sensitive data is inadvertently shared on the global server, the parameter aggregation strategy in ALMITY, which does not necessitate any sensitive information from datasets, ensures that this mistakenly uploaded information remains unused, is not preserved, and is promptly eliminated before the next round of data uploading. Moreover, ALMITY offers users the flexibility to freely define the minimum number of participating clients required for ALMITY. Consequently, in situations where certain clients encounter communication issues with the server, as long as the number of clients engaged in regular communication exceeds the predefined threshold, the model will continue to function seamlessly.

## IX. THREATS TO VALIDITY

We identify the following threats to the validity of our study:

**Threats to external validity** primarily pertain to the quality and generalizability of ALMITY. Our study involves evaluations on two prominent SE tasks: code clone detection using the

BigCloneBench [44] dataset and software defect prediction utilizing data from *Devign* [35]. Considering that ALMITY is a task-agnostic framework, conducting additional case studies on diverse datasets can further validate the generalizability of our findings and the applicability of our model, ALMITY, across different scenarios.

**Threats to internal validity** primarily relate to potential errors in our model selection and parameter settings. We opt for CodeBERT [37] as our pre-trained model, given its extensive use in prior SE research [40] and its typical high performance in downstream tasks [35], [57], [58]. While our results have demonstrated strong performance with our framework, ALMITY, future research could potentially enhance the model by exploring alternative pre-trained models. Additionally, different parameters, such as the number of clients used within ALMITY, can influence the effectiveness of the trained models. To investigate the impact of these choices, we have conducted experiments where we systematically increased or decreased the number of clients, including 2, 3, 4, 5, and 6 clients, in ALMITY framework (refer to Table I and Table II).

**Threats to construct validity** primarily concern the possibility of human errors. We have taken measures to mitigate these threats by selecting highly experienced researchers to participate in the user study and instructing them to independently respond to the questions presented in our user study.

## X. RELATED WORK

In this section, we will explain prior research in three aspects: 1) federated learning in software engineering, 2) code clone detection, and 3) software defect prediction.

**Federated Learning in Software Engineering.** Extensive prior research [21], [59] has been proposed to study the application on federated learning, e.g., healthcare [60], [61], finance [62], [63], and internet of things [64], [65]. Nonetheless, only a small number of research works examine the application of federated learning in software engineering. Abyane et al. [66] conduct an empirical study to investigate the quality and robustness of federated learning from multiple angles of attacks. Motivated by the fact that web-services structure can provide a means for robust integration of distributed data, Verma et al. [67] implement a web service based federated learning framework on enterprises using large-scale distributed data. Additionally, the existing related works are conducted or implemented in the use of direct federated learning frameworks. Note that data in different organizations may suffer from heterogeneous and imbalanced issues. Our study is the first work on the use of federated learning on traditional software engineering tasks while addressing the imbalanced data issue.

**Code Clone Detection.** Code clone detection is a typical software engineering task. Much prior research has been proposed to detect code clones [68], [69]. Typically, these techniques can be broken into five categories, Text-based, Token-based, AST-based, Graph-based, and metric-based techniques. Text-based techniques are first proposed to detect clones. In the Text-based technique, the source code of a software system is considered a sequence of raw strings. Many text-based

clone detection techniques and tools, e.g., SDD [70], have been proposed to detect clones. Token-based techniques use lexical tools to parse source programs into a sequence of lexical tokens. CCFinder [71], and SourcererCC [72] are two typical token-based clone detectors. AST-based techniques often apply different parsers to transform source programs into abstract syntax trees (AST). Deckard [4], and NiCad [73] are the most popular AST-based clone detectors. Graph-based approaches parse programs into a graph such as data flow and control flow, to represent source code. Duplix [74] and GPLAG [75] are two typical examples of graph-based tools. Metric-based techniques extract metrics from direct source code or tree and graph representation of source code. Oreo [76] and eMetrics [77] are two metrics-based clone detectors. Recently, Liu et al. [78] find that the lack of clone data of which the functionalities have never been previously observed in the training dataset leads to low performance of deep learning-based clone detectors. This finding highlights the need for more real-world clone data to improve the performance of clone detectors.

**Software Defect Prediction.** There exists a large body of research aiming to predict software defects using product and process metrics to build a model. The first work on software defect prediction is proposed by Mockus and Weiss [79]. Mockus and Weiss build a linear regression model to predict the probability of failure after code changes using change metrics including code size, duration, diffusion, as well as the experience of the developers. The discovered failure-inducing changes may be incomplete. Therefore, Kamei et al. [80], [81], [82] perform a series of works on commit-level defect prediction considering more change metrics. Various kinds of complicated models, e.g., Naive Bayes [6], support vector machines [83], random forest [84], and deep learning-based models [85], have been also applied for predicting software defects. Different from the above research, our goal is not to create a novel prediction model. Our work proposes a federated learning-based framework ALMITY to help and improve the performance of software defect prediction.

## XI. CONCLUSION AND FUTURE WORK

To facilitate the comprehensive utilization of industrial-sensitive data in bridging the gap between academic SE research and industry applications, we introduce a collaborative paradigm called ALMITY. It empowers sensitive datasets to be fully harnessed for training highly effective learning-based models even on skewed data distributions, all without the need to address data privacy and security concerns. ALMITY is built upon federated learning principles and enhances the parameter aggregation strategy by incorporating three key attributes: data scale, data balance, and minority class learnability. This design is aimed at addressing the issue of low model performance when trained on real-world skewed datasets. To assess the effectiveness and generalizability of ALMITY, we conduct extensive comparative experiments to evaluate the performance of ML and DL models trained by different training methods. We consider two commonly used training methods as baselines: centralized training method (CTM) and vanilla federated

learning method (VFL). These experiments are conducted within the context of two well-studied SE tasks, i.e., the code clone detection task and the defect prediction task. Furthermore, to assess the practical applicability of ALMITY, we curate a dataset collection encompassing diverse data distributions derived from real-world task-specific scenarios. We proceed to evaluate ALMITY's performance across all types of data distributions within this dataset collection. The experimental results demonstrate that employing ALMITY enables learning-based models, i.e., ML and DL models, to attain superior performance on both academic and skewed datasets compared to baseline methods. Our future work entails broadening our evaluation scope to encompass additional SE tasks.

## ACKNOWLEDGMENT

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

## REFERENCES

- [1] K. Bjerger, H. M. Mann, and T. T. Høye, "Real-time insect tracking and monitoring with computer vision and deep learning," *Remote Sens. Ecology Conservation*, vol. 8, no. 3, pp. 315–327, 2022.
- [2] A. S. Subramanian, C. Weng, S. Watanabe, M. Yu, and D. Yu, "Deep learning based multi-source localization with source splitting and its effectiveness in multi-talker speech recognition," *Comput. Speech Lang.*, vol. 75, 2022, Art. no. 101360.
- [3] I. Lauriola, A. Lavelli, and F. Aioli, "An introduction to deep learning in natural language processing: Models, techniques, and tools," *Neurocomputing*, vol. 470, pp. 443–456, Jan. 2022.
- [4] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and accurate tree-based detection of code clones," in *Proc. 29th Int. Conf. Softw. Eng. (ICSE)*, Minneapolis, MN, USA, Los Alamitos, CA, USA: IEEE Comput. Soc. Press, May 20–26, 2007, pp. 96–105.
- [5] Y. Wu et al., "SCDetector: Software functional clone detection based on semantic tokens analysis," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2020, pp. 821–833.
- [6] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 309–320.
- [7] AIMultiple. Accessed: Jun. 5, 2023. [Online]. Available: <https://research.aimultiple.com/large-language-model-evaluation/>
- [8] B. Koch, E. Denton, A. Hanna, and J. G. Foster, "Reduced, reused and recycled: The life of a dataset in machine learning research," 2021, *arXiv:2112.01716*.
- [9] A. R. Munappy, J. Bosch, H. H. Olsson, A. Arpteg, and B. Brinne, "Data management for production quality deep learning models: Challenges and solutions," *J. Syst. Softw.*, 2022, Art. no. 111359.
- [10] Y. M. Herrera and D. Kapur, "Improving data quality: Actors, incentives, and capabilities," *Political Anal.*, vol. 15, no. 4, pp. 365–386, 2007.
- [11] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-IID data silos: An experimental study," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 965–978.
- [12] W. Rahman et al., "Clone detection on large Scala codebases," in *Proc. IEEE 14th Int. Workshop Softw. Clones (IWSC)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 38–44.
- [13] Unily. Accessed: 2023. [Online]. Available: <https://www.unily.com/privacy-hub/security-certifications>
- [14] Anonymous GitHub. Accessed: Dec. 26, 2023. [Online]. Available: <https://anonymous.4open.science/t/ALMITY-20B5>
- [15] J. R. Cordy and C. K. Roy, "The NiCad clone detector," in *Proc. IEEE 19th Int. Conf. Program Comprehension*, Piscataway, NJ, USA: IEEE Press, 2011, pp. 219–220.
- [16] H. Sajjani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: Scaling code clone detection to big-code," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 1157–1168.



- [17] C. J. Kasper and M. W. Godfrey, "‘Cloning considered harmful’ considered harmful: Patterns of cloning in software," *Empirical Softw. Eng.*, vol. 13, no. 6, pp. 645–692, 2008.
- [18] M. R. Islam, M. F. Zibran, and A. Nagpal, "Security vulnerabilities in categories of clones and non-cloned code: An empirical study," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 20–29.
- [19] F. Learning, "Collaborative machine learning without centralized training data," 2017. Available: <https://blog.research.google/2017/04/federated-learning-collaborative.html>
- [20] P. P. Liang et al., "Think locally, act globally: Federated learning with local and global representations," 2020, *arXiv:2001.01523*.
- [21] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.
- [22] S. K. Lo, Q. Lu, C. Wang, H.-Y. Paik, and L. Zhu, "A systematic literature review on federated machine learning: From a software engineering perspective," *ACM Comput. Surv. (CSUR)*, vol. 54, no. 5, pp. 1–39, 2021.
- [23] H. H. Zhuo, W. Feng, Y. Lin, Q. Xu, and Q. Yang, "Federated deep reinforcement learning," 2019, *arXiv:1901.08277*.
- [24] H. Yu et al., "A fairness-aware incentive scheme for federated learning," in *Proc. AAAI/ACM Conf. AI, Ethics, Soc.*, 2020, pp. 393–399.
- [25] S. Truex et al., "A hybrid approach to privacy-preserving federated learning," in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, 2019, pp. 1–11.
- [26] A. A. Süzen and M. A. Şimşek, "A novel approach to machine learning application to protection privacy data in healthcare: Federated learning," *Namik Kemal Tip Dergisi*, vol. 8, no. 1, pp. 22–30, 2020.
- [27] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, PMLR, 2017, pp. 1273–1282.
- [28] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, 2021.
- [29] Wikipedia. Accessed: Dec. 18, 2023. [Online]. Available: [https://en.wikipedia.org/wiki/Entropy\\_\(information\\_theory\)](https://en.wikipedia.org/wiki/Entropy_(information_theory))
- [30] Stack Exchange Inc. Accessed: Oct. 13, 2016. [Online]. Available: <https://stats.stackexchange.com/questions/239973/a-general-measure-of-data-set-imbalance>
- [31] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilingual token-based code clone detection system for large scale source code," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, Jul. 2002.
- [32] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 87–98.
- [33] H. Wei and M. Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code," in *Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2017, pp. 3034–3040.
- [34] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, "Deep learning similarities from different representations of source code," in *Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories (MSR)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 542–553.
- [35] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 10197–10207.
- [36] C. Ni, X. Xia, D. Lo, X. Yang, A. E. Hassan, "Just-in-time defect prediction on JavaScript projects: A replication study," *ACM Trans. Softw. Eng. Method.*, vol. 31, no. 4, pp. 1–38, 2022.
- [37] Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," 2020, *arXiv:2002.08155*.
- [38] D. Guo et al., "GraphCodeBERT: Pre-training code representations with data flow," 2020, *arXiv:2009.08366*.
- [39] J. A. Harer et al., "Automated software vulnerability detection with machine learning," 2018, *arXiv:1803.04497*.
- [40] S. Lu et al., "CodeXGLUE: A machine learning benchmark dataset for code understanding and generation," 2021, *arXiv:2102.04664*.
- [41] H. A. Alhija, M. Azzeh, and F. Almasalha, "Software defect prediction using support vector machine," 2022, *arXiv:2209.14299*.
- [42] S. Jadon, "Code clones detection using machine learning technique: Support vector machine," in *Proc. Int. Conf. Comput., Commun. Automat. (ICCCA)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 399–303.
- [43] GitHub. Accessed: Mar. 21, 2020. [Online]. Available: <https://github.com/feiwu/PROMISE-backup/tree/master>
- [44] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, "Towards a big data curated benchmark of inter-project code clones," in *Proc. IEEE Int. Conf. Softw. Maintenance Evolution*, Piscataway, NJ, USA: IEEE Press, 2014, pp. 476–480.
- [45] GitHub. Accessed: May 1, 2022. [Online]. Available: <https://github.com/Jur1cek/gcj-dataset>
- [46] G. Zhao and J. Huang, "DeepSim: Deep learning code functional similarity," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 141–151.
- [47] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FEDAVG on non-IID data," 2019, *arXiv:1907.02189*.
- [48] V. Arammongkolvichai, R. Koschke, C. Ragkhitwetsagul, M. Choetkieritikul, and T. Sunetnanta, "Improving clone detection precision using machine learning techniques," in *Proc. 10th Int. Workshop Empirical Softw. Eng. Pract. (IWESEP)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 31–315.
- [49] J. Zhou et al., "Finding a needle in a haystack: Automated mining of silent vulnerability fixes," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 705–716.
- [50] J. Carka, M. Esposito, and D. Falessi, "On effort-aware metrics for defect prediction," *Empirical Softw. Eng.*, vol. 27, no. 6, 2022, Art. no. 152.
- [51] M. Yan, X. Xia, E. Shihab, D. Lo, J. Yin, and X. Yang, "Automating change-level self-admitted technical debt determination," *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1211–1229, Dec. 2019.
- [52] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.
- [53] A. Luque, A. Carrasco, A. Martín, and A. de Las Heras, "The impact of class imbalance in classification performance metrics based on the binary confusion matrix," *Pattern Recognit.*, vol. 91, pp. 216–231, Jul. 2019.
- [54] Wikipedia. Accessed: Aug. 24, 2023. [Online]. Available: [https://en.wikipedia.org/wiki/Wilcoxon\\_signed-rank\\_test](https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test)
- [55] R. Malhotra, A. Budhiraja, A. K. Singh, I. Ghoshal, and S. Meena, "Multiple feature selection frameworks based on evolutionary computing and ensemble learning for software defect prediction," in *Proc. 5th Int. Conf. Comput. Intell. Netw. (CINE)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 1–6.
- [56] R. Moussa and F. Sarro, "On the use of evaluation measures for defect prediction studies," in *Proc. 31st ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2022, pp. 101–113.
- [57] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evolution Reeng. (SANER)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 261–271.
- [58] V. Raychev, P. Bielik, and M. Vechev, "Probabilistic model for code with decision trees," *ACM SIGPLAN Notices*, vol. 51, no. 10, pp. 731–747, 2016.
- [59] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol. (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.
- [60] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *Int. J. Med. Informatics*, vol. 112, pp. 59–67, Apr. 2018.
- [61] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *J. Healthcare Inform. Res.*, vol. 5, no. 1, pp. 1–19, 2021.
- [62] W. Yang, Y. Zhang, K. Ye, L. Li, and C.-Z. Xu, "FFD: A federated learning based method for credit card fraud detection," in *Proc. Int. Conf. Big Data*, New York, NY, USA: Springer-Verlag, 2019, pp. 18–32.
- [63] G. Long, Y. Tan, J. Jiang, and C. Zhang, "Federated learning for open banking," in *Federated Learning*. New York, NY, USA: Springer-Verlag, 2020, pp. 240–254.
- [64] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for Internet of Things: Recent advances, taxonomy, and open

- challenges,” *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1759–1799, Third Quart. 2021.
- [65] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, “Federated learning for Internet of Things: A comprehensive survey,” *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1622–1658, Third Quart. 2021.
- [66] A. E. Abyane, D. Zhu, R. M. de Souza, L. Ma, and H. Hemmati, “Towards understanding quality challenges of the federated learning: A first look from the lens of robustness,” 2022, *arXiv:2201.01409*.
- [67] D. C. Verma, G. White, and G. de Mel, “Federated AI for the enterprise: A web services based implementation,” in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Milan, Italy, Jul. 8–13, 2019, Piscataway, NJ, USA: IEEE Press, 2019, pp. 20–27.
- [68] C. K. Roy, J. R. Cordy, and R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” *Sci. Comput. Program.*, vol. 74, no. 7, pp. 470–495, 2009.
- [69] D. Rattan, R. K. Bhatia, and M. Singh, “Software clone detection: A systematic review,” *Inf. Softw. Technol.*, vol. 55, no. 7, pp. 1165–1199, 2013.
- [70] S. Lee and I. Jeong, “SDD: High performance code clone detection system for large scale source code,” in *Proc. Companion 20th Annu. ACM SIGPLAN Conf. Object-Oriented Program., Syst., Lang., Appl. (OOPSLA)*, San Diego, CA, USA. New York, NY, USA: ACM, Oct. 16–20, 2005, pp. 140–141.
- [71] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A multilingual token-based code clone detection system for large scale source code,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, Jul. 2002.
- [72] H. Sajjani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, “SourcererCC: Scaling code clone detection to big-code,” in *Proc. 38th Int. Conf. Softw. Eng. (ICSE)*, Austin, TX, USA. New York, NY, USA: ACM, May 14–22, 2016, pp. 1157–1168.
- [73] C. K. Roy and J. R. Cordy, “NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization,” in *Proc. 16th IEEE Int. Conf. Program Comprehension (ICPC)*, Amsterdam, The Netherlands. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Jun. 10–13, 2008, pp. 172–181.
- [74] J. Krinke, “Identifying similar code with program dependence graphs,” in *Proc. 8th Work. Conf. Reverse Eng. (WCRE)*, Stuttgart, Germany. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Oct. 2–5, 2001, pp. 301–309.
- [75] C. Liu, C. Chen, J. Han, and P. S. Yu, “GPLAG: Detection of software plagiarism by program dependence graph analysis,” in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Philadelphia, PA, USA. New York, NY, USA: ACM, Aug. 20–23, 2006, pp. 872–881.
- [76] V. Saini, F. Farmahinifarahani, Y. Lu, P. Baldi, and C. V. Lopes, “Oreo: Detection of clones in the twilight zone,” in *Proc. ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng., (ESEC/SIGSOFT FSE)*, Lake Buena Vista, FL, USA. New York, NY, USA: ACM, Nov. 4–9, 2018, pp. 354–365.
- [77] F. Lanubile and T. Mallardo, “Finding function clones in web applications,” in *Proc. 7th Eur. Conf. Softw. Maintenance Reeng. (CSMR)*, Benevento, Italy. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Mar. 26–28, 2003, p. 379.
- [78] C. Liu, Z. Lin, J.-G. Lou, L. Wen, and D. Zhang, “Can neural clone detection generalize to unseen functionalities  $f$ ,” in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 617–629.
- [79] A. Mockus and D. M. Weiss, “Predicting risk of software changes,” *Bell Labs Tech. J.*, vol. 5, no. 2, pp. 169–180, 2000.
- [80] Y. Kamei et al., “A large-scale empirical study of just-in-time quality assurance,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2013.
- [81] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, “Studying just-in-time defect prediction using cross-project models,” *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 2072–2106, 2016.
- [82] Y. Kamei and E. Shihab, “Defect prediction: Accomplishments and future challenges,” in *Proc. IEEE 23rd Conf. Softw. Anal., Evolution, Reeng. (SANER)*, vol. 5, Piscataway, NJ, USA: IEEE Press, 2016, pp. 33–45.
- [83] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, “HYDRA: Massively compositional model for cross-project defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 42, no. 10, pp. 977–998, 2016.
- [84] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, “Software defect prediction using feature selection and random forest algorithm,” in *Proc. Int. Conf. New Trends Comput. Sci. (ICTCS)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 252–257.
- [85] J. Li, P. He, J. Zhu, and M. R. Lyu, “Software defect prediction via convolutional neural network,” in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 318–328.