Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

FedCSD: A Federated Learning Based Approach for Code-Smell Detection

SADI ALAWADI[1], KHALID ALKHARABSHEH [2], FAHED ALKHABBAS [3,4], VICTOR R. KEBANDE (MEMBER, IEEE) [1], FERAS M. AWAYSHEH[5], FABIO PALOMBA [6], MOHAMMED AWAD[7]

[1]Department of Computer Science, Blekinge Institute of Technology, Karlskrona, Sweden ({sadi.alawadi,victor.kebande}@bth.se)
[2]Software Engineering Department, Prince Abdullah bin Ghazi Faculty of Information and Communication Technology, Al-Balqa Applied University, As-Salt, Jordan. (khalidkh@bau.edu.jo)
[3]Internet of Things and People Research Center, Malmö University, Malmö, Sweden. (fahed.alkhabbas@mau.se)
[4]Department of Computer Science and Media Technology, Malmö University, Malmö, Sweden. (fahed.alkhabbas@mau.se)
[5]Institute of Computer Science, Delta Research Centre, University of Tartu, Tartu, Estonia. (feras.awaysheh@ut.ee)
[6]University of Salerno, Italy. (fpalomba@unisa.it)
[7]Arab American University, Palestine. (mohammed.awad@aaup.edu)

Corresponding author: Khalid Alkharabsheh (e-mail: khalidkh@bau.edu.jo).

ABSTRACT

Software quality is critical, as low quality, or "Code smell," increases technical debt and maintenance costs. There is a timely need for a collaborative model that detects and manages code smells by learning from diverse and distributed data sources while respecting privacy and providing a scalable solution for continuously integrating new patterns and practices in code quality management. However, the current literature is still missing such capabilities. This paper addresses the previous challenges by proposing a Federated Learning Code Smell Detection (FedCSD) approach, specifically targeting "God Class," to enable organizations to train distributed ML models while safeguarding data privacy collaboratively. We conduct experiments using manually validated datasets to detect and analyze code smell scenarios to validate our approach. Experiment 1, a centralized training experiment, revealed varying accuracies across datasets, with dataset two achieving the lowest accuracy (92.30%) and datasets one and three achieving the highest (98.90% and 99.5%, respectively). Experiment 2, focusing on cross-evaluation, showed a significant drop in accuracy (lowest: 63.80%) when fewer smells were present in the training dataset, reflecting technical debt. Experiment 3 involved splitting the dataset across 10 companies, resulting in a global model accuracy of 98.34%, comparable to the centralized model's highest accuracy. The application of federated ML techniques demonstrates promising performance improvements in code-smell detection, benefiting both software developers and researchers.

INDEX TERMS Software Quality, Technical Debit, Federated Learning, Privacy-preserving, Code Smell Detection.

I. INTRODUCTION

Software quality assurance is a major aspect that occupies the minds of software engineers and the software engineering community at large. Consequently, there is a continuous need to maintain the quality of the software, given that it is a determinant in many aspects during and after development. Specifically, software quality assurance determines and detects the software pieces that suffer from low quality in design or programming. These pieces are known as "Code Smells" [1]. The existence of code smells does not produce errors during compilation or execution [2] , they also negatively influence the software quality factors [3]–[5].

Consequently, the availability of the code smells increases the time and effort required to maintain the software. This extra time and effort is known as technical debt [6], which can be indicated by the presence of code smells. Several terms and concepts have been used to denote code smells, such as antipatterns, disharmonies, design flaws, design defects, code anomalies, design smells, etc [4]. Code smells can be identified in various software components, from instructions to subsystems, and can influence different levels of software granularity, such as methods, classes, and the whole system.

Code smell detection is an efficient way to decrease maintenance costs and support the efforts of software developers to improve the quality of software. Due to the increasing size and complexity of the developed software systems, more automated approaches are needed to improve the activity of code smell detection.

At present, several approaches concentrate on code smell detection, such as metric/rule-based approaches [7]–[12] and machine learning-based approaches [13]–[19]. Most of these approaches have been evaluated empirically, and they have obtained high precision in smell detection. However, there are a set of challenges that constrain their endorsement in the industry, such as the ratio of false negatives and false positives in their findings and the low degree of agreement between them.

To overcome the existing shortcomings and challenges, literature studies, [15]–[17], [20]–[22], have shown that machine learning-based approaches play a central role in code smell detection and can be more exploited in this direction. Consequently, it is possible to make a quantum leap in improving the detection of the right code smells with high accuracy. ML uses mathematical algorithms to award systems the ability to learn without explicitly programming [23].

The use of centralized ML training to detect code smells has been widely investigated. For instance, in [24]–[26], the authors compared the performance of multiple ML algorithms for code smell severity detection over different datasets. The centralized training process demands a considerable amount of collected and aggregated data, typically in a centralized place such as a data centre, cloud, or server machine. This centralized data aggregation is imperative for constructing an accurate model with quality that adapts to dynamic data, aiming to provide recommendations, decisions, and solutions for specific tasks. However, the considerable expense associated with transferring data to a central hub presents a significant hurdle. Moreover, this data often contains sensitive and private information belonging to the data owner, leading to concerns regarding both data privacy and security. Such matters counter General Data Protection Regulation (GDPR) policies and pose challenges across various sectors, including healthcare, industry, politics, etc.

As a concrete example, let's examine the software industry within the context of our research. In this industry, every company holds a significant stake in understanding the source code and design quality employed by their competitors. Aspects such as performance, maintainability, and reusability are of utmost importance. These companies are keen on leveraging this invaluable data to elevate the quality of their software products. They aim to identify and rectify anomalies and deficiencies in their codebase while refining coding practices and policies to produce top-notch enterprise software. However, it's crucial to note that no company within this competitive landscape will release their private data, including their source code and design details. Instead, they are in pursuit of techniques that allow them to extract valuable insights and knowledge from other companies' data in a secure and non-invasive manner.

In this regard, Federated Learning (FL) emerges as an efficient solution that maintains data privacy and security. FL differs from centralized ML in migrating the ML model to the data's source for training, typically on the edge side [27]. Unlike centralized ML, FL enables all edge-node models to contribute their knowledge without exposing the raw data (source code or design in our case). By employing FL, software enterprises that are hesitant to share their data can internally train their ML models and then transfer the learned model to a designated entity responsible for maintaining software quality [28], [29].

The main contribution of this paper is to propose Federated Learning Code Smell Detection (FedCSD), which, to the best of our knowledge, is the first approach that exploits FL for code smell detection. Specifically, the God Class smell. We show how FedCSD can be applied in settings where multiple software development companies collaborate to improve the quality of their software development projects without the need to share their code. Further, we discuss how FedCSD can improve the traditional code review activity within software development teams. Finally, we present intensive experiments that show the advantages of applying FedCSD to detect code smells in comparison to traditional centralized ML approaches.

The remainder of this paper is organized as follows. Section II introduces background on code smell detection tools, the role of machine learning in code smell detection, federated learning, and data privacy and attacks. Additionally, it discusses related studies. Section III presents the methodology we applied to design and validate our approach. Section IV describes the proposed approach. Next, Section V analyses and discusses the results. while Section VI discusses the critical evaluation of the study. Finally, Section VII presents the threats of validity and Section VIII presents conclusions and recommendations for future work.

II. BACKGROUND AND RELATED WORK

A. CODE SMELL DETECTION TOOLS

Several code smell detection tools have been developed either as standalone or integrated, commercial or open-source, and they support different programming languages and detect various types of code smells. Examples of these tools include iPlasma, jCosmo, Incode, DECOR, PMD, Borland Together, and JDeodorant [4], [30]. However, they have limitations that reduce their effectiveness in industry. Namely, they have a low degree of agreement, lack the capability to analyze software systems implemented in more than one programming language, do not detect a wide set of different types of code smells, lack the interoperability of detection tools with diverse development environments [31], and how scalable code smell detection techniques are to large-scale codebases.

One of the code smells that detection tools focus on the most is Large Class [1], also referred to as God Class [32], and the Blob [5]. In the literature, several studies have

focused on detecting the Large Class code smell [33]–[41]. In their systematic mapping study, the authors [4] analyzed close to 400 articles related to code smells and found that Large Class negatively affects different software quality attributes, the most important of which is maintainability. Based on the above and since Large Class is one of the code smells most frequently detected in software systems, we decided to focus on it in this work.

B. MACHINE LEARNING IN CODE SMELL DETECTION

In one study [16], ML and object-oriented metrics extracted from analyzing software systems were combined into an approach to automatically detect design flaws. The proposed approach was evaluated on three open-source systems. The findings showed that the decision tree effectively detects Large Class and Long Method smells. Another study [17] utilized ML to predict seven types of design smells (Message Chains, Middle Man, Switch Statement, Long Parameter List, Long Method, Feature Envy, and Lazy Class). The dataset was constructed from a group of 27 metrics gathered from software systems, including design smells.

Furthermore, in [15], the Bayesian Detection Expert (BDTEX) approach was proposed to detect well-known antipatterns named Functional Decomposition, Spaghetti Code, and the Blob. The approach was evaluated on two systems, and the results were compared with the DECOR approach. Further, one study [42] presented a novel approach that uses the support vector machine and object-oriented metrics to detect Swiss Army Knife, the Blob, Spaghetti Code, and Functional Decomposition antipatterns by analyzing three software systems. The results were compared with the DETEX approach. In another work by [43], five ML techniques were used based on software metrics to detect different antipatterns. The presented approach was named NiPAD. The study was conducted using one application, and the result showed that the best behavior was obtained by the SVMlinear technique for identifying the One-lane Bridge antipattern.

A more recent research study by [22] used 16 ML classifiers to detect Data Class, God Class, Long Method, and Feature Envy code smells. The chosen smells were automatically detected using five tools. The study evaluated 74 software systems, and the results of detection were validated manually by experts in the domain. The findings showed that most of the techniques have a high degree of accuracy. Moreover, in [19], the performance of metric-based and machine learning-based approaches was empirically compared in terms of code smell detection. The dataset was constructed from 13 software systems and 17 metrics to detect 11 code smells. The results showed that metric-based approaches achieve slightly better performance. Nonetheless, there is a need to conduct more studies on both approaches in order to enhance the precision and efficiency of code smell detection. Recently, the authors of [37] conducted a large-scale study that investigated the usefulness of ML techniques for effective design smell detection. The work focused on determining the influence of data balancing on the accuracy of ML techniques during design smell detection. A set of 28 classifiers was used to detect God Class design smells in a dataset of 24 software systems that include 12,587 classes, and the detection results were validated manually by experts. After replicating the experiments on two more datasets, the findings showed there is no significant influence of data balancing on the accuracy of learning classifiers during design smell detection. Moreover, machine learning approaches are efficient in God Class detection. Detecting SQL code smells using code analysis seems like interesting future work [44].

All the above studies concluded that standard machine learning-based approaches have a promising and efficient role in the code smell detection context. However, it has some limitations concerning the obtained model. The generated model has been trained on a dataset stored in a centralized place and gathered from different open source software projects located in well-known repositories. In this case, due to the data privacy risks concerning data leaks or misuse and the reluctance of companies to share their complete project data on these repositories, there might be a lack of information about the project context that should be taken into account when training the model, such as architectural patterns, domain-specific requirements, and coding conventions. Therefore, the model may not completely comprehend the intricacies of each software project's coding practises. Consequentially, the model's accuracy will be affected. In this work, to overcome the limitations of previous works, we exploited the advantages of federated learning for code smell detection. On the one hand, our approach involves significant project-specific context information that can be lacking or cannot be shared between companies when training the model, resulting in a more accurate and generalizable code smell detection model. On the other hand, our approach preserves better the data privacy and security of software projects, as companies do not need to share their code repositories.

C. FEDERATED LEARNING

Big data systems [45] and traditional ML approaches are centralized approaches. In general, they require data to be collected and aggregated offline on one site, where the models are trained and deployed [46], [47]. These approaches have some shortcomings for code smell detection because training and deploying ML models in central nodes requires companies to disclose the source code of their projects. Similarly, distributed learning approaches require the code to be released to the distributed servers. Thus, such approaches do not address the companies' privacy concerns [48], [49].

To overcome the aforementioned limitations, Google proposed FL, an emerging paradigm that enables users or organizations to jointly train an ML model without releasing their private data [47], [50], [51]. Fl follows the privacy-by-design philosophy [52]. Specifically, in FL settings, companies can train a global ML model to detect code smells collaboratively by aggregating the local trained models' updated parameters (gradients, weights) and reporting them to the FL server,

where the global model will be constructed, then propagating the global model to all the involved companies or contributors. Therefore, by exploiting FL, the companies do not need to share their source code or data, thereby preserving their privacy, and they just share their models' learned knowledge that collaboratively builds a comprehensive global model to detect code smells.

D. DATA PRIVACY AND SECURITY THREATS

Scenarios surrounding traditional ML allude to the fact that centralization plays a major role in holding training data and executing the learning algorithm. In real world situations,legal restrictions and privacy laws prohibit sharing even well-trained models across diverse participants [53]. Data being the driving force behind the running of many companies, it is worth noting that a majority of ML models are owned by tech companies; based on how these companies manage the data and code behind the running of operations, this raises pertinent questions of centralization [54]. Centralization in this context provides an avenue for a single point of failure and increases the threat levels and potential attack surfaces with the possibility of zero day vulnerabilities. To accommodate data owners' constant need for secure and collaborative execution of data, an FL environment provides guarantees and assurances from a privacy and security perspective. However, the literature pinpoints other diverse scenarios that emphasizes privacy, for example, during model-training [55] and also during security-based model training [56].

Privacy concerns in code smell detection are important in the software development landscape. This importance is mainly seen when we consider the inherent sensitivity of both code and data ownership. In the conventional paradigm of centralized code smell detection, where code repositories are extensively analyzed, a critical issue emerges where potential exposure of sensitive or proprietary information is imminent. These repositories often hold the confidential data that forms the backbone of an organization's software projects. When sharing such code repositories or data with external parties, traditional methods inadvertently raise significant privacy concerns. In this study, we assess these capabilities from the perspective of intentional or unintentional data leakage, which could compromise the trained model [56].

By harnessing the power of FL, it is envisaged that organizations could collectively train machine learning models while keeping their code and data firmly within their own walls. This collaborative yet privacy-preserving approach ensures that sensitive information remains confidential and proprietary algorithms stay safeguarded.

Security-related threats in the code smell could pose significant challenges to the integrity and reliability of the code quality assessment process. The authors from key assumptions like code injection on original code and adversarial attacks are prevalent in many cases when preparing, training, or deploying learning models. The common form of security related attacks involves adversarial manipulation, tampering with code and training data during the testing phases, indiscriminate attacks where an adversary makes wrong decisions in order to damage the classifiers, integrity attacks, and availability attacks focused on degrading the usability of the FL system and the code deployed by increasing the positive rate [57]–[59].

These unauthorized infiltrations can lead to false positives or negatives in code smell detection, rendering the entire process unreliable. In traditional centralized approaches, where code repositories are shared, the risk of such attacks is heightened, as external access to code repositories becomes a potential point of entry for adversaries.

The suggestions on possible defense strategies that preserve privacy and allow open and closed code to withstand these attacks are discussed in the subsequent sections of this paper.

E. WHY FL IN CSD?

The integration of FL in CSD represents a significant paradigm shift, addressing several limitations inherent in traditional CSD methodologies (See Table 1). FL's decentralized nature fundamentally enhances data privacy and security, a critical concern in software development where codebases often contain sensitive or proprietary information. By processing data locally at the node level, FL circumvents the need to centralize sensitive code, thus preserving confidentiality while enabling practical code analysis. Moreover, FL's handling of diverse and distributed data sources is particularly advantageous in CSD. Traditional approaches typically rely on centralized data aggregation, which needs to improve with the size and diversity of code repositories. However, FL excels at learning from heterogeneous data sources, offering a more robust and inclusive code quality analysis. This scalability is further beneficial in large-scale projects or organizations where FL distributes computational load across multiple nodes, mitigating resource constraints centralized systems face.

However, the implementation of FL in CSD has its challenges (See Table 1). Ensuring robust local data processing while managing data diversity and consistency across various nodes introduces complexity. Another significant challenge is addressing data sparsity and imbalance without introducing biases, especially with non-iid datasets [60]. Moreover, when detecting rare or subtle code smells. Despite these challenges, FL's ability to provide real-time, decentralized, and privacy-preserving analysis and its scalability and adaptability to diverse datasets make it an effective and valuable method for CSD. The approach enhances the integrity and reliability of code quality assessment and aligns with the evolving needs of modern, distributed software development practices. Thus, while the path to seamlessly integrating FL in CSD involves navigating certain complexities, the overarching advantages it presents in terms of security, scalability, and comprehensive analysis make it a compelling approach in the realm of code quality management.

Challenge in CSD	**Traditional Solutions**	**FL Solution**	**FL Contribution to CSD**	**FL Challenges**
Data Privacy and Security	Anonymization (reduces data utility)	Local data processing enhances privacy.	Secure CSD without compromising code.	Ensuring robust local processing, handling data diversity.
Diverse and Distributed Data Sources	Centralized data aggregation (size and diversity issues)	Learns effectively from diverse, decentralized sources.	Accurate CSD in varied environments.	Managing data consistency across nodes.
Scalability and Resource Constraints	Powerful centralized servers (costly, less scalable)	Scalable, distributed computational load.	Scalable CSD in large organizations.	Balancing load, ensuring node processing power.
Handling Imbalanced and Sparse Data	Oversampling (introduces biases)	Manages imbalanced data sets by learning from various nodes.	Improved CSD for rare or sparse smells.	Addressing data sparsity and imbalance without biasing.
Integrity and Reliability of Code Quality Assessment	Manual reviews, simplified tools (potential biases)	Decentralized validation for unbiased code quality assessment.	Reliable, unbiased code quality assessment.	Ensuring robustness and impartiality in decentralized validation.

TABLE 1: Challenges in Code Smell Detection and Federated Learning opportunities.

TABLE 2: Goal of this study

Purpose	Analyze
Object	set of classes
With the purpose of	Evaluation
With respect to	the efficiency of federated learning approach to detect God class code smell in different scenarios
Viewpoint	from researchers and practitioners point of view
In the context of	closed-source software companies

III. RESEARCH METHODOLOGY

To address the challenges highlighted in Section II and better answer of the needs of the software development communities, we propose the FedCSD approach. For this purpose, we applied in an iterative way the design science research method, as we aimed at devising an innovative approach that solves a practical problem and this is supported by the selected methodology. Specifically, we followed the well-defined guidelines for conducting design science research, which comprises five stages, namely, problem explication, requirements definition, artifact design and development, artifact demonstration, and artifact evaluation [61].

In the *problem explication* stage, we reviewed the literature for code smell detection approaches. We found that no studies investigated the use of FL for code smell detection purposes. Accordingly, we used the Goal-Question-Metric (GQM) approach, which is commonly used by the software engineering community [62], [63], to formulate our study goal presented in Table 2.

In the *requirements definition* stage, we defined one requirement based on the defined goal, namely, to devise an approach that exploits FL to detect code smells cross-organizations. Accordingly, we formulated the following research questions and hypotheses:

RQ1 *How can federated learning be effectively leveraged for God Class code smell detection?*
Objective: by answering this RQ, we aim to understand how FL can be applied within each software development company and also across different companies to detect code smells and consequently improve the software systems' quality.

RQ2 *How does the use of federated learning affect the quality of the resulting ML model compared to the individual models generated by centralized training approaches?*
Objective: by answering this RQ, we aim at comparing the performances of FL and centralized ML models in detecting code smells.

The null hypotheses have been formulated as follows:

Hypothesis 1: *Federated learning cannot be effectively leveraged for God Class code smell detection.*

Hypothesis 2: *Federated learning does not improve the quality of the resulting ML model compared to the individual models generated by centralized training approaches.*

In the *artifact design and development* stage, we proposed the first approach that exploits FL to detect code smells during the software development phase (see Section IV). Specifically, our approach shows how multiple organizations can collaboratively exploit FL to train ML models without the need to share their code and use the models to improve the qualities of their code. Further, our approach evolves the traditional code review life cycle by integrating the models trained collaboratively.

In the *artifact demonstration and evaluation* stages, we simulated how our approach can be applied in cross-organizational settings and ran experiments that validate its feasibility, respectively (see Section V).

IV. FEDCSD APPROACH

This section presents our approach for Federated Learning Code Smell Detection (FedCSD). To the best of our knowledge, FedCSD is the first proposed approach that evolves the traditional code review life cycle by integrating FL to detect code smells during development. Consequently, our approach

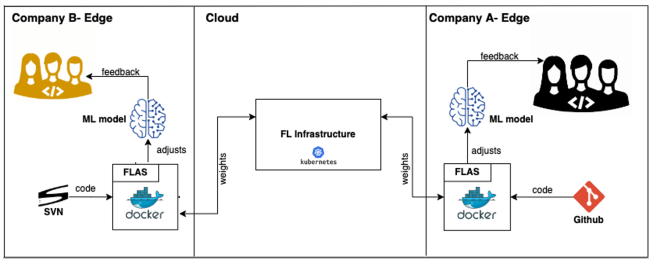

FIGURE 1: Federated Learning in cross-organization settings

enables addressing code smells proactively, unlike the majority of existing approaches, which manage technical debt issues reactively. The approach exploits traditional architectures of FL and is mainly based on the FEDn framework.

First, we introduce how the approach supports multiple organizations to employ FL to collaboratively train ML models that can detect code smells with higher accuracy. Then, we describe how the FedCSD evolves the traditional code review activity within the software development life cycle.

A. FEDCSD IN ACTION

Figure 1 shows abstractly how FedCSD supports organizations to exploit FL to train ML models and use them to improve the quality of their code. For this purpose, companies need to set up FedCSD containers, which can be realized using Dockers [1].

For this purpose, companies need to link the containers with both their code repositories and the data pipeline that will be used to train the ML model locally in FL settings. The containers run the client endpoint that automatically starts the local training rounds (i.e., on the edge of the network [64]), reports the updated weights' resulting from the local training to the FL aggregators, and updates the ML models' weights according to the results of the global model parameters constructed by the FL aggregator (reducer or server). The FL aggregators (reducers and combiners) run on a shared cloud environment that auto-scales based on the number of involved clients using the Kubernetes [2] technology. Algorithm 1 shows the FedAvg algorithm used to derive global weights using the weights generated during local training rounds. The models provide feedback to the developers about their code quality, as described in detail below.

The FL infrastructure of FedCSD adopts the FEDn federated learning framework [65]. FEDn is an open-source framework that follows the hierarchical MapReduce paradigm. Figure 2 illustrates FEDn architecture composed of three layers: reducers, combiners, and clients.

The reducer acts as a server in the server-client paradigm, which has several responsibilities, including the following: (1) monitoring the model training; (2) controlling the communications flow among all the federation components; (3) initiating the seed model with a random weight and then distributing it among the connected combiners; (4) propagating the computing package where the model training and validation instructions are descried to combiners, then from combiners to the connected clients; (5) starting the training (i.e., communication rounds); and (6) aggregating all the updated parameters of the combiners' local–global models and then averaging them using the FedAvg algorithm (see Algorithm 1) to construct the final global model. Meanwhile, the combiner represents the intermediate layer responsible for the following: (1) linking the reducer with different client nodes to decrease the reducer's computation load and the network communication workload; (2) distributing the received model from the reducer across all corresponding clients; and (3) combining all local models' updated gradients provided by the connected clients using the FedAvg algorithm to build the local–global model.

Finally, the client layer represents the companies' local servers (edge nodes), where the data is placed and the local model training rounds are performed. Each client in the federation will receive from the combiner both the ML model and the computing package, which is considered the guideline for the client to train the model. Algorithm 2 explains the training process in the client's local node per communication round. The algorithm returns the model's updated parameters, which will be reported backward to the upper layer. Each client should be connected to a combiner, but multiple clients can be connected to the same combiner, as can be seen in Figure 2.

Algorithm 1: FedAvg algorithm, where **k** is the number of clients, **r** is the number of rounds, W_i is the local model weights and **M** is the global model weights

Input: W_t
Output: $M(W_t)$

Server executes:
initialized W_0
Function `FedAVG` (k, W_{t-1}, W_t) **:**
  foreach $t \leftarrow 1$ **to** r **do**
    $S_t \leftarrow$ (sample a random set of clients)
    foreach *client* $k \in S_t$ ***in parallel*** **do**
      $W_{t+1}^k \leftarrow ClientUpdate(k, W_t, N_l)$
      $W_{t+1} \leftarrow \sum_{k=1}^{k} \frac{n_k}{n} W_{t+1}^k$
    end
    $W_t \leftarrow (W_{t-1} + (W_t - W_{t-1})/t)$
  end
  return $M(W_t)$

B. EVOLVED CODE-REVIEW ACTIVITY

Code review is one of the main activities that reduces maintenance costs and technical debt. Detecting and addressing code smells early prevents them from becoming more com-

[1] https://www.docker.com
[2] https://kubernetes.io/

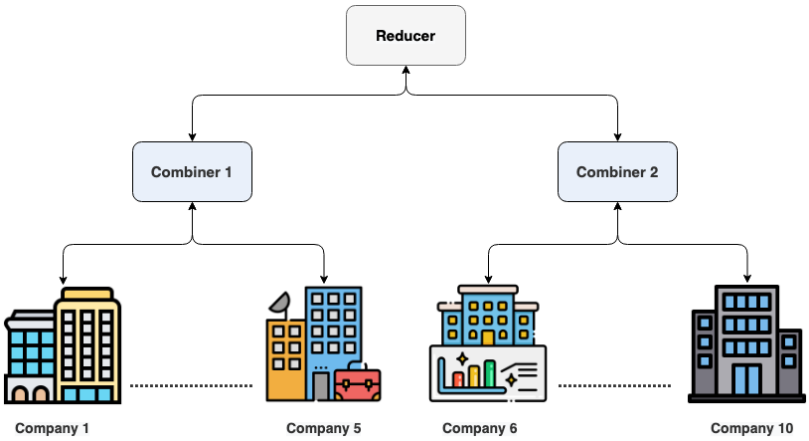

FIGURE 2: A high level representation of FedCSD's main components, including 1 reducer, 2 combiners, and 10 companies

Algorithm 2: Local client update, where **k** is the number of clients, $\boldsymbol{D^k}$ is client k local dataset, **e** is the number of local epochs, and η is the learning rate

Output: W_t

/* Run on client k */

Function ClientUpdate(k, W_t)**:**
  $\beta \leftarrow$ (split D^k into mini batches)
  for *local epoch* $e_i \in 1, \ldots e$ **do**
    for *batch* $b \in \beta$ **do**
      $W_t \leftarrow W_t - \eta \nabla l(W_t, b)$
    end
  end
  return W_t

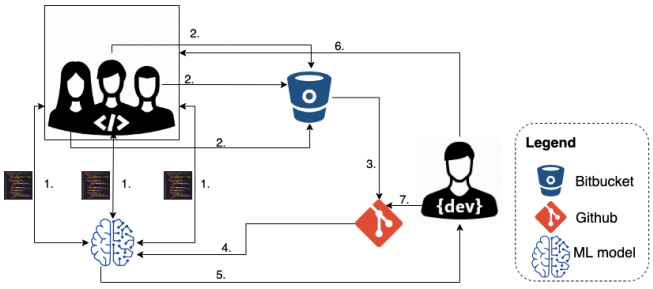

FIGURE 3: The evolved code review cycle

plex problems in the future, especially when the software systems are large-scale. Also, developers can improve their knowledge and learning experience by discussing and determining refactoring opportunities suggested by ML models. Also, thanks to FL, our approach enables the generation of ML models that are trained on a variety of software systems from multiple companies without the need to share the code bases of the different projects.

Figure 3 shows the code-review activity, which is part of the traditional software development life cycle, evolved by exploiting FL to improve the code quality. The cycle starts when engineers or developers request to check their individual code for smells before creating code review requests to their team leads or other experienced team members (**step 1**). Then, the ML model detects smells in the developers' individual code and provides them with feedback (**step 2**). Accordingly, the developers update their individual code and then create code review requests (e.g., using Bitbucket) (**step 3**). After that, the team members responsible for reviewing the different code submitted for review receive feedback from the ML model about smells detected in the entire code being reviewed (**steps 4 and 5**). Accordingly, the code reviewers provide feedback, possibly to multiple developers, to adjust the code to improve its quality (**step 6**). This cycle continues until all the comments on the code are addressed and no smells are detected. Consequently, the code submitted for review is approved and merged to the suitable branch (e.g., a release branch on GitHub) (**step 7**).

V. RESULTS AND DISCUSSION

To validate and evaluate the feasibility of the proposed approach in code smell detection activity, we designed the following experiments:

1) **Experiment 1**: Train the code smell detection model for each dataset centrally and evaluate its performance in terms of accuracy.
2) **Experiment 2**: Simulate a new coding behavior scenario for a real company by evaluating the trained ML model using parts of other datasets (cross-validation).
3) **Experiment 3**: Evaluate our approach by splitting the three datasets into different chunks to simulate 10 distinct companies that will participate in training the global ML model.

The code used to run the experiments is available via Github [3].

[3]https://github.com/saadiabadi/codeSmill.git

A. EXPERIMENTAL SETTINGS

In this section, we describe the datasets used in this study. After that, we describe the Long Short-Term Memory (LSTM) algorithm to automatically detect the god class code smell.

1) Dataset

To examine the proposed approach, we used three datasets from the literature: [18], [66], [67]. The details of these datasets concerning the number of classes, the number of methods, and the total lines of code in each software are shown in Tables 3, 4, and 5, respectively, while the comparisons among them are shown in Table 6. The 1st (**Pecorelli et al.**) and 2nd (**Fontana et al.**) datasets were constructed by [18] and [67], respectively, whereas the 3rd (**Khalid et al.**) was collected by our team. The set of software systems used in each dataset were open-source, written in Java, came from different domains and size categories, and were available in different repositories, such as Github and SourceForge. Moreover, they are well-known and widely used in the code smell detection context. Concerning (our) dataset (i.e., the 3rd one), we followed concrete criteria to collect the software systems from the repositories, including the number of downloads, availability in several versions, and the history of software systems' maintenance. Due to the huge number of systems that met the criteria, we randomly selected twenty four systems. The datasets focused on detecting different types of code smells, and the God Class was one of them. According to Fowler [1], a large class is a class that tries to do too many tasks, making it very large regarding the total number of lines of code, number of methods, number of variables, and dependencies with other classes. Therefore, the possibility of duplicate code will increase. Moreover, this class has high complexity as well as low cohesion. [5]. Table 6 presents the characteristics of the chosen datasets in terms of the number of projects, the number of classes, the number of detection tools, and the number of detected God Classes (GC) using tools (GC-Tool), the number of human experts who participated in the manual validation process, and the number of God Classes detected by experts (GC-Experts). The total number of software systems was 111 and was formed of more than $80,000$ classes. Each dataset was analyzed automatically by a set of detection tools, and the detection results were manually validated by a group of human experts who have good knowledge of code smell detection. The results of the manual validation were formulated as a binary decision (God Class = 1, Not God Class = 0). As a result, the number of false positives God Classes was reduced in all datasets from $2,696$ to 721, which represents a 26% reduction. To meet the objective of the study, we preprocessed all datasets to have both the same features and format. Table 7 reports the 16 features of the dataset, their definitions, and the quality dimensions of different software levels. The replication package in [68] includes all the datasets.

TABLE 3: Pecorelli et al. dataset characteristics.

Project	NOC	NOM	TLOC
ant-rel-1.8.3	1,473	13,213	119,256
argouml-VERSION_0_14	1,373	9,045	199,075
cassandra-cassandra-1.1.0	699	11,360	110,712
apache-wicket-1.4.11	1,568	12,429	174,033
derby-10.3.3.0	1,746	5,987	535,187
hadoop-release-0.2.0	327	2,460	34,662
hsqldb-2.2.0	590	5,004	254,014
incubator-livy-0.6.0-incubating	1,016	450	130,696
nutch-release-0.7	532	3,220	50,578
qpid-0.18	2,172	21,448	189,271
xerces-Xerces-J_1_4_2	489	6,088	150,445
eclipse-R3_4	5,061	924	423,423
elasticsearch-v0.19.0	1,395	21,739	315,619

2) Long Short-Term Memory (LSTM) Algorithm

We used the Long Short-Term Memory (LSTM) model to detect the God Class code smell over the datasets mentioned earlier. The structure of LSTM [69] depends on three gates, an input gate, a memory and forgetting gate, and an output gate. The input gate regulates the flow of information, the forget gate ensures that unimportant information is forgotten, Ft is used to refer to the following mechanism.

$$F_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (1)$$

The i_t represents the input gate that is used to retain the neural network's state and to determine which data will be incorporated into the cell's state

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (2)$$

The output gate O_t presents what extent and how information is filtered out of the neural network.

$$O_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3)$$

Where σ is the activation function, W_f, W_i , and W_o are the weights value, h_{t-1}, is output value before 't', x_t, is input value at 't', and b_t, b_i , and b_o are the bias value for the 3 gates.

The model was implemented using the public TensorFlow framework implementation from Keras [4]. The model architecture is composed of one LSTM input layer with 16 dimensions, four dense layers with 72, 50, 36, and 28 units, and a ReLU activation function. The output layer distinguishes between God Class and Not God Class. For instance, we fixed the model hyper-parameters per communication round to have a 0.001 initial learning rate with the Adam optimizer. Then, we fixed the batch size to 32. Finally, we set the maximum number of epochs to 1. This model has been used in both centralized and collaborative training experiments (FedCSD).

Further, we simulated 10 different companies using the datasets mentioned earlier by using the Pecorelli et al. [18] dataset, partitioned to five chunks; the Fontana et al. [67] dataset, not partitioned; and the Khalid et al. [37] dataset, partitioned to 4 chunks. In total, all of the previous data

[4]https://github.com/tensorflow/tensorflow

TABLE 4: Fontana et al. dataset characteristics.

Project	NOC	NOM	TLOC
aoi-2.8.1	799	688	136,533
argouml-0.34	2,361	18,015	284,934
axion-1.0.M2	223	2,989	28,583
castor-1.3.1	1,542	11,967	213,479
cobertura-1.9.4.1	107	3309	58,364
colt-1.2.0	525	4,143	75,688
columba-1.0	1,188	6818	109,035
displaytag-1.2	128	1,064	20,892
drawswf-1.2.9	297	2,742	38,451
drjava-20100913-r5387	225	10,364	130,132
emma-2.0.5312	262	1,805	34,404
exoportal-1.0.2	1,855	11,709	102,803
findbugs-1.3.9	1,631	10,153	146,551
fitjava-1.0.1	60	254	2,453
fitlibraryforfitnesse-20100806	795	4,165	25,691
freecol-0.10.3	1,244	8,322	163,595
freecs-1.3.20100406	131	1,404	25,747
freemind-0.9.0	849	5,788	65,687
galleon-2.3.0	764	4,305	12,072
ganttproject-2.0.9	959	5,518	58,718
heritrix-1.14.4	649	5,366	9,424
hsqldb-2.0.0	465	7,652	171,667
itext-5.0.3	497	5,768	117,757
jag-6.1	255	145	24,112
jasml-0.10	48	524	6,694
jasperreports-3.7.3	1,571	17,113	260,912
javacc-5.0	102	808	19,045
jedit4.3.2	1,037	656	138,536
jena-2.6.3	1,196	99	117,117
jext-5.0	485	2,169	34,855
jFin_DateMath-1.0.1	58	541	7,842
jfreechart-1.0.13	960	1,181	247,421
jgraph-5.13.0	399	2,996	53,577
jgraphpad-5.10.0.2	426	1,879	33,431
jgrapht-0.8.1	299	1,475	28,493
jgroups-2.10.0	1,093	8,798	126,255
jhotdraw-7.5.1	968	7,232	104,357
jmeter-2.5.1	909	8,059	113,375
jmoney-0.4.4	190	713	9,457
jparse-0.96	65	780	16,524
jpf-1.0.2	121	1,271	18,172
jruby-1.5.2	2,023	17,693	199,533
jspwiki-2.8.4	405	2,714	69,144
jsXe-04_beta	100	703	1,448
jung-2.0.1	786	3,884	53,617
junit-4.1	204	1,031	9,065
log4j-1.2.16	296	2,118	34,617
lucene-3.5.0	1,908	12,486	214,819
marauroa-3.8.1	208	1,593	26,472
megamek-0.35.18	2,096	13,676	315,953
mvnforum-1.2.2-ga	338	5,983	92,696
nekohtml-1.9.14	56	502	10,835
openjms-0.7.7-beta-1	515	379	68,929
oscache-2.4.1	66	629	11,929
picocontainer-2.10.2	208	1,302	12,103
pmd-4.2.5	862	5,959	71,486
poi-3.6	233	19,618	299,402
pooka-3.0-080505	813	68,127	68,127
proguard-4.5.1	604	5,154	82,661
quartz-1.8.3	280	2,923	52,319
quickserver-1.4.7	132	1,278	18,243
quilt-0.6-a-5	66	641	8,425
roller-4.0.1	567	5,715	78,591
squirrel_sql-3.1.2	153	689	8,378
sunflow-0.07.2	191	1,447	24,319
tomcat-7.0.2	1,538	15,627	283,829
trove-2.1.0	91	585	8,432
velocity-1.6.4	388	2,957	5,559
wct-1.5.2	606	5,527	69,698
webmail-0.7.10	118	1,092	14,175
Weka-3.7.5	2,045	17,321	390,008
xalan-2.7.1	1,171	10,384	312,068
xerces-2.10.0	789	9,246	188,289
xmojo-5.0.0	110	1,199	31,037

TABLE 5: Khalid et al. dataset characteristics.

Project	NOC	NOM	TLOC
JCLEC-4-base	311	1,647	37,575
FullSync-0.10.2	169	1,467	24,323
AngryIPScanner-3.0	270	1,228	19,965
SQuirreL-1.2	1,138	19,031	71,626
Javagraphplan-1.0	50	537	1,049
DigiExtractor-2.5.2	80	523	15,668
JFreechart-1.0.X	499	8,024	206,559
Plugfy-0.6	28	103	2,337
sMeta-1.0.3	222	1,912	30,843
Ganttproject-2.0.10	621	5,047	66,540
xena-6.1.0	1,975	1,272	61,526
pmd-4.3.x	800	6,021	82,885
JDistlib-0.3.8	78	1,027	32,081
Matte-1.7	603	4,170	52,067
JasperReports-4.7.1	1,797	18,781	350,690
Mpxj-4.7	553	11,634	261,971
Apeiron-2.92	62	702	8,908
OmegaT-3.1.8	716	5,115	121,909
Lucene-3.0.0	606	12,459	81,611
KeyStoreExplorer-5.1	384	2,535	83,144
Freemind-1.0.1	782	6,824	106,396
heckstyle-6.2.0	277	606	41,104
jAudio-1.0.4	416	4,799	117,615
JHotDraw-5.2	151	1,497	17,807

TABLE 6: Characteristics of the datasets used in this study

Dataset	#Project	#Class	#Tool	#GC-Tools	#Experts	GC-Experts
1st (**Pecorelli et al.** [18])	13	18,441	1	318	2	96
2nd (**Fontana et al.** [67])	74	55,000	2	420	3	140
3rd (**Khalid et al.** [66])	24	12,587	5	1,958	3	485

TABLE 7: Dataset features.

No.	Feature	Definition	Granularity	Dimension
1	TLOC	Total Lines of Code	Project	Size
2	NCLOC	Non-Comment Lines of Code	Project	Size
3	CLOC	Comment Lines of Code	Project	Size
4	EXEC	Executable Statements	Project	Complexity
5	DC	Density of Comments	Project	Complexity
6	NOT	Number of Types	Package	Complexity
7	NOTa	Number of Abstract Types	Package	Complexity
8	NOTc	Number of Concrete Types	Package	Complexity
9	NOTe	Number of Exported Types	Package	Complexity
10	RFC	Response for Class	Class	Coupling
11	WMC	Weighted Methods per Class	Class	Complexity
12	DIT	Depth in Tree	Class	Inheritance
13	NOC	Number of Children in Tree	Class	Inheritance
14	DIP	Dependency Inversion Principle	Class	Coupling
15	LCOM	Lack of Cohesion of Methods	Class	Cohesion
16	NOA	Number of Attributes	Class	Size

chunks represent 10 different heterogeneous companies, as shown in Figure 2 in the client layer. Moreover, we relayed our experiment over Swedish National Infrastructure for Computing (SNIC) Science Cloud [70] resources, and all instances used in the experiment have 8 Virtual Centralized Processing Units (VCPU), and 16GB RAM.

B. EVALUATION METRICS

To evaluate the FedCSD approach, we used the evaluation metrics Accuracy, Loss Function, Kappa, and ROC Area, which are well-known in the literature for evaluating ML in code smell detection. Each metric evaluates the performance of the proposed approach from a different aspect.

- Accuracy represents the ratio of correctly classified samples (true positive and true negative). In this study, it is the percentage of classes that are predicted correctly as God Class/Not God Class. However, the accuracy value falls between 0 and 100 and can be computed using Equation 4. Higher values indicate a more accurate prediction.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} * 100\% \quad (4)$$

- Loss Function is a method to evaluate ML algorithms concerning how well the obtained model is qualified to predict the expected classification results. If the predicted results are distant from the actual results, the value of the loss function will be high. This value denotes the errors in the prediction process and can be reduced through learning the loss function. We used the categorical cross-entropy as a loss function, as shown in Equation 5.

$$Loss = -\sum_{i=1}^{N} y_i . \log \hat{y}_i \quad (5)$$

where $\hat{y}_i$ is the model prediction for *i-th* pattern, y_i represent the corresponding real value, and *N* is the total number of samples.

- Cohen Kappa is a test that assesses the concordance between the samples that ML algorithms classified and the actually labelled data. The values of the Kappa measure range from -1 to 1, where the higher value denotes a strong degree of concordance. The Cohen Kappa can be computed using Equation 6.

$$\kappa = \frac{P_o - P_e}{1 - P_e}. \quad (6)$$

Where P_o represents the samples ratio agreement, and P_e shows the expected agreement percentage between samples. Moreover, the interpretation of the kappa values is shown in Table 8.

TABLE 8: Kappa Values Interpretation.

Kappa	Agreement
$kappa < 0.20$	Poor
$0.21 \leq kappa < 0.40$	Fair
$0.41 \leq kappa < 0.60$	Moderate
$0.61 \leq kappa < 0.80$	Substantial
$0.81 \leq kappa \leq 1.00$	Almost perfect

- The ROC-Area, the area under the Response Operating Characteristic (ROC) curve, is a well-known test that is used to evaluate, organize, and visualize the effectiveness of ML algorithms. It focuses on identifying the relationships between the specificity and sensitivity of learning algorithms. The ROC values range from 0 to 1. The higher ROC value indicates a better learning model. Table 9 presents the interpretation of the test.

TABLE 9: ROC area Interpretation.

Value	Interpretation
$0.5 < ROC \leq 0.6$	Fail
$0.6 < ROC \leq 0.7$	Poor
$0.7 < ROC \leq 0.8$	Fair
$0.8 < ROC \leq 0.9$	Good
$0.9 < ROC \leq 1$	Excellent

C. EXPERIMENT 1: CENTRALIZED TRAINING

In this experiment, a centralized ML model was trained over each mentioned dataset to evaluate the performance of the code smell detection model in a traditional company scenario. Table 10 reports the model's accuracy per dataset. We noticed that the model has achieved high accuracy over all datasets. For instance, training the ML model over both **Khalid et al.** and **Pecorelli et al.** datasets has obtained higher performance, with a slight difference related to the number of smells covered by each dataset. Meanwhile, with the **Fontana et al.**dataset, the model obtained the lowest accuracy (92.30%), detecting fewer smells than what actually exists. The nature of the dataset, such as software quality, size, and diversity, plays a main role in the model's accuracy. The number of projects used in each dataset as well as the number of classes were different and belonged to different size categories (large, medium, etc.). Therefore, there are differences in the size of the training dataset used to train the model, which directly influence the model's accuracy, as shown in the cases of **Pecorelli et al.** and **Khalid et al.**, which were larger than the **Fontana et al.** dataset. In addition, the set of software projects came from various software domains (application, development, etc.) and statuses (stable, mature, etc.) and were randomly included in the datasets. All these factors influence the model's accuracy and should be taken into account when producing robust and accurate detection models. Therefore, we hypothesized that any change in the company's coding culture or new workers joining the company with different coding behaviour would affect the ML model's performance and cause technical debt.

TABLE 10: The accuracy achieved by the centralized ML model per dataset

Dataset	Model Accuracy
First dataset (**Pecorelli et al.**)	98.90%
Second Dataset (**Fontana et al.**)	92.30%
Third dataset (**Khalid et al.**)	99.15%

D. EXPERIMENT 2: ML MODEL CROSS-EVALUATION

Changing the company's coding culture will likely introduce divergences in model performance, leading to concept drift that can adversely influence the model's outputs. To evaluate the resilience and robustness of the centrally trained model against such shifts, we have simulated new changes in companies' coding behaviour that could appear from a new team member or updates in company policies that significantly impact both their internal culture and their products. Therefore,

we trained the ML model over one dataset and validated it over the other two datasets (cross-evaluation). Table 11 reports the accuracy achieved by the model trained based on earlier settings. Notably, when the model was trained on either **Pecorelli et al.** or **Khalid et al.** and tested on the other, we noticed a small gap in model accuracy compared with the results obtained in experiment 1 (see V-C) for both datasets, and the model has achieved the highest accuracy (96.30% and 97.00%, respectively) using both datasets in this context, which refers to the fact that both companies shared the same coding behaviour and culture.

In contrast, when training or testing the ML model on the **Fontana et al.** dataset, a notable and significant drop in model accuracy was observed across all cases between 15% and 30%. The model obtained the lowest accuracy (63.80%) compared to the highest accuracy (97.00%) achieved using other datasets. It is essential to highlight that the **Fontana et al.** dataset covers different types of smells with further distribution as other datasets. Furthermore, the drift concept, whether in terms of data or model drift, introduces an additional dimension to the context, highlighting the critical steps needed to improve model performance and adaptability in such scenarios.

In conclusion, this experiment clearly shows the significant impact of coding behaviour or culture changes on the smell detection model. By comparing the results obtained from experiments 1 and 2, we observed examples of the model drift concept affecting performance significantly. At the same time, in other cases, the impact was relatively insignificant. To tackle this problem, we propose our FedCSD approach, where the global model is collaboratively trained and built, leveraging contributions from various companies. This approach effectively captures and adapts to changes in a company's culture and its team's coding behaviour.

TABLE 11: Trained LSTM model evaluated over the other two datasets in a centralized fashion.

		Testing dataset		
		First dataset (**Pecorelli et al.**)	Second Dataset (**Fontana et al.**)	Third dataset (**Khalid et al.**)
Training dataset	First dataset (**Pecorelli et al.**)		63.80%	96.30%
	Second Dataset (**Fontana et al.**)	79.00%		80.00%
	Third dataset (**Khalid et al.**)	97.00%	71.00%	

E. EXPERIMENT 3: FEDCSD EVALUATION

We conducted this experiment to answer the research questions presented in Section **??** as well as to validate our proposed approach (FedCSD) in terms of global model performance (Accuracy and Loss), prediction agreement (Kappa), sensitivity and specificity (ROC value). In this experiment, we simulated 10 companies to participate in the federation by splitting both the **Pecorelli et al.** and **Khalid et al.** datasets into chunks that represent five and four companies, respectively, and keeping the **Fontana et al.** dataset to represent one company. This allowed us to maintain all clients' heterogeneity and replicate a real scenario. Consequently, the experiment showed the power of federated learning and mitigated the challenges we faced in the previous experiments. Further, we found that this will reduce the computation cost by leveraging the edge nodes (companies) resources to train the model, preserve each company's data privacy, construct a global model that has a comprehensive knowledge of code smells accumulated from all clients, and reduce the opportunity of having technical debt if new smells appear.

Figure 4 shows the model's loss function behavior over the testing set for 100 rounds, which is generally employed over the training and validation sets to optimize the ML algorithm. This metric was calculated using the model prediction for every sample and its corresponding actual output individually, indicating how bad or good the model is. Figure 4 shows the improvement of the model's learning process after each training round. The testing showed that the model behaved perfectly after round 40, which indicates that the model had reached optimal behavior based on the loss value.

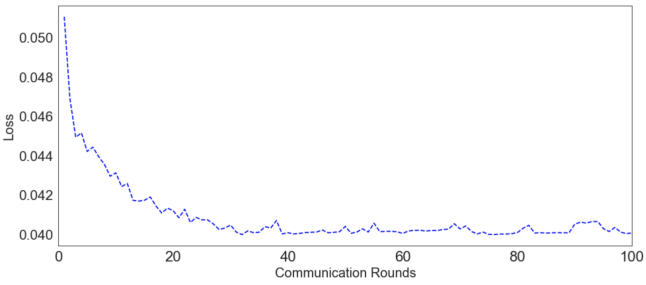

FIGURE 4: Global federated learning model loss function of 10 different clients (companies) for 100 communication rounds

After obtaining the optimal optimization of the model using the loss function, the model's performance was also evaluated in terms of accuracy. Figure 5 illustrates the global model's performance for 100 training rounds. We noticed that the initial model accuracy was high (97.7%) and very close to the centralized results. Moreover, the global model's performance converged in a considerable direction and reached 98.34%. We noticed that around round 63, the model started to stabilize with only a slight oscillation (0.04%) until round 95.

Comparing our FedCSD accuracy with experiments 1 and 2, we argue that our model outperforms both the centralized and cross-evaluation experiments, despite the FedCSD accuracy being a bit lower than the results obtained from the model trained over the Khalid et al. dataset (see Table 10) by almost 0.1%. This difference can be ignored in favor of both the model global knowledge and the model stability provided by the FedCSD.

Evaluating our FedCSD approach in terms of the agreement ratio between the global model prediction and the corresponding actual value demonstrates the robustness of our approach. Figure 6 depicts the Cohen Kappa measured

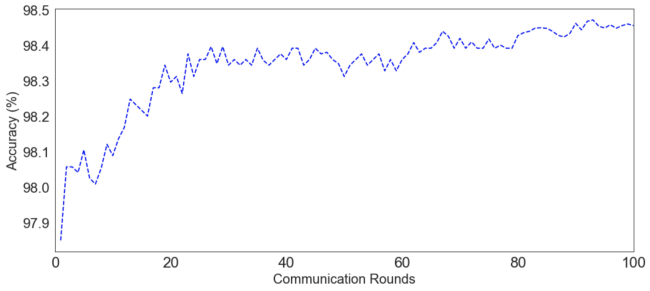

FIGURE 5: Global federated learning model accuracy of 10 different clients (companies) for 100 communication rounds

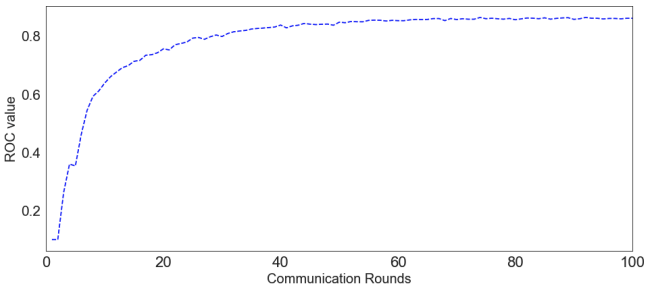

FIGURE 7: Global federated learning model ROC value of 10 clients (companies) for 100 communication rounds

while testing the global model for 100 training rounds. There is a significant improvement in the Kappa value per training round, where the initial agreement was low (around 55%) then linearly converged in the right direction. Further, the Kappa value started to stabilize after round 60 and obtained 79%, which falls in the substantial range as indicated in Table 8. Therefore, the acquired agreement ratio shows the strong learning ability of the FedCSD, which can be generalized for code smell detection problems.

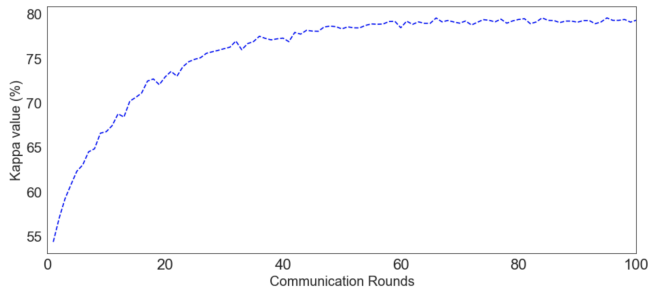

FIGURE 6: Global federated learning model kappa value (%) of 10 clients (companies) for 100 communication rounds

In addition to the previous metrics, we calculated the ROC value for the global model constructed by our approach during 100 rounds. The ROC value is an essential and accurate metric used to evaluate classification problems that do not rely on class distributions. As shown in Figure 7, the ROC value curve depicts the trade-off between sensitivity (Y-axis) and specificity (X-axis). However, the global model improved linearly per training round, similarly to the previous metrics. Moreover, Figure 7 highlights how after round 60, the model stabilized without any anomalous behavior, which guides us to the conclusion that our approach can capture or learn any new change in the coding culture.

Based on the above, in general, we find that the ML model trained based on the proposed approach (FedCSD) has achieved high performance values according to the loss function, accuracy, kappa, and ROC measurements when detecting code smell. As a result, we conclude that the federated learning approach can effectively be leveraged for God Class code smell detection. Both null hypotheses are therefore rejected.

VI. DISCUSSION: CRITICAL EVALUATION OF THE STUDY

The experiments that have been conducted in this study have shown the importance of applying FL to preserve privacy during the training of an ML model, owing to the complexity that is involved during software design. While code smells are seen as perennial issues that affect the quality of software, machine learning, specifically the federated aspect, is touted as a game changer in code smell detection, not only with a higher degree of accuracy but also with precision in preserving essential attributes of the code and data. Accordingly, this study offers an optimal approach aimed at addressing the endemic and perennial privacy issues prevalent during the ML training phase. We are aware of the fact that the knowledge extracted during data training plays a significant role; hence, securely distributing and showing the extracted knowledge from the FL approach emerges as an effective solution to the challenges posed by traditional centralized machine learning methods. Furthermore, leveraging data from open-source software in FedCSD offers considerable advantages, particularly in code smell detection.

Notably, this approach can be applied in a multitude organizations that will collaboratively be able to not only train ML models but also detect code smells, preserve the privacy of their data, and at the same time expose the relevant security-related risks during the training of ML models. The significant findings of this study can be summarized as follows:

The implications of our findings extend beyond the confines of this research and are addressed as follows:

- **Collaborative Software Quality Enhancement:** Our study demonstrates that organisations can effectively harness FL to train and enhance machine learning models collaboratively. This approach holds profound significance for improving the quality of software code while preserving the essential privacy of data and code. Practitioners in software development can leverage this approach to detect and rectify code smells proactively, thus minimizing technical debt and enhancing software maintainability. Therefore, the FedCSD approach introduces a valuable user-feedback component, allowing developers to assess their code quality and detect

code smells collaboratively for continuous improvement. This interactive aspect fosters a culture of code quality awareness within development teams, leading to better code practices and enhanced software quality.

- **Privacy-Preserving Software Development:** The study addresses the critical challenge of privacy in code smell detection, a concern that has often been/not been extensively explored based on existing literature. Our findings emphasize the importance of privacy preservation in machine learning applications and provide a blueprint for other domains where sensitive data is involved.
- **On-the-fly Code Smell Detection :** The validity of the experiments that were conducted in this study shows that code detection models aid in not only detecting software design flaws but also improving accuracy during this discourse. The experiments conducted in this study validate the efficacy of code detection models in not only identifying software design flaws but also improving overall accuracy. This holds great promise for practitioners seeking robust tools to assess and enhance their code quality.

In addition, we have explored the need for exploring potential security vulnerabilities and adversarial learning threats, which, as a result of implementing FedCSD, may affect integrity and privacy. Firstly, it is pertinent to explore the existence of attack vectors [71], which have far-reaching implications for the deployment of FedCSD. For example, aspects like model poisoning [72], where malicious participants may inject erroneous data during training holds. Others include data poisoning [73], model and gradient inversion attacks [74], and membership inference attacks [75]. The aforementioned are key adversarial techniques that may defeat the FedCSD implementation. While the scope of this research does not go into an exploration of key mitigation strategies, from a privacy perspective, we identify leveraging differential privacy [76], secure aggregation, model encryption, and the use of robust federated optimization algorithms that are essential for ensuring the effectiveness and privacy of federated learning-based approaches such as FedCSD.

We argue that leveraging FL to address privacy related aspects in code smell detection is innovative given that, at the time of writing this paper, there was a gap in research on code smell detection using closed datasets, improving accuracy, and involving multiple organizations while also simultaneously preventing code exposure. Importantly, this study outlines the shortcomings of traditional ML, which, from a privacy and security perspective, increases the threat and attack levels of the learning models. While we acknowledge that the study does not go into detail in identifying specific security attacks, it is worth mentioning that we have taken a step in highlighting the generic security-related aspects that could be of interest when deploying FedCSD. Given that the scope of the study is not majorly inclined towards security, we were not concerned with the significance of this aspect during ML phases (as pointed out by [57]–[59]). However, we consider this an avenue for future work.

While this study primarily focuses on code smell detection and privacy preservation, it opens up several avenues for future research. Specifically, there is potential for further exploration of security aspects, such as specific security attacks during machine learning phases, as well as the development of more sophisticated privacy-preserving techniques within the Federated Learning framework.

Ultimately, our study bridges the gap in research related to code smell detection using closed datasets, accuracy improvement, and multi-organizational collaboration while maintaining code confidentiality, as was seen in the scenario that was leveraged in this study. We acknowledge the limitations of not delving into specific security attacks, and we recognize this as an area ripe for future investigation. The implications of this research extend to practitioners who seek to elevate software quality while safeguarding data privacy and security, making it not only a significant contribution to academia but also a valuable resource for industry professionals.

Further, we have taken a positive step in acknowledging the previous related studies that have not only laid a firm foundation for this work but have also provided key insights that have significantly consolidated the arguments put forth in this paper.

VII. THREATS TO VALIDITY

This section presents the various threats to the validity of our proposed approach.

A. CONSTRUCT VALIDITY

Construct validity concerns the tools and algorithms exploited for code smell detection purposes. Accordingly, one threat concerns the use of the *Fedn* framework. In [65], the authors conducted multiple experiments that validated the Fedn framework's scalability, resource utilization, and training accuracy. Another threat to validity concerns the use of the LSTM algorithm. When we performed the experiments, the Fedn framework supported only deep learning algorithms. We chose the LSTM algorithm because it is known for its ability to store information from previous steps and use that information to influence the output of the current step. Additionally, the LSTM achieved almost the same score as the best code smell detection algorithm reported in [37].

B. INTERNAL VALIDITY

An internal threat to the validity of our approach concerns the distribution of the used datasets with respect to the class instances (i.e., god class/not god class). Unbalanced datasets can affect the quality of the trained ML model. To mitigate this threat, we applied oversampling and undersampling techniques in order to balance the three datasets used in the experiments.

C. EXTERNAL VALIDITY

An external threat to the validity of our approach concerns the generalization of our experiments' results. Specifically, application of our approach for detecting code smells, e.g., in commercial and/or non open-source software systems. To mitigate this threat, we considered three datasets of open source systems with different application domains and size categories. Indeed, our experiments show that the performance of the ML trained using the datasets outperforms the performance of a centralized ML model trained on an individual dataset.

VIII. CONCLUSIONS AND FUTURE WORKS

Great strides have been made in developing federated learning as a distributed AI-based technique as far as the enhancement of privacy of data is concerned. However, at the time of writing this paper, there existed limited or no research that leveraged federated learning in not only detecting code smells but also preserving privacy at the same time. As a result, the research that has been reported in this paper has explored a privacy-aware approach by proposing a Federated Learning Code Smell Detection (FedCSD) that is significant for organizations. The relevance of this proposition is that it enables organizations to ensure software quality and preserve the privacy of their data at the same time by mainly sharing only knowledge as opposed to data. Specifically, in this paper, we demonstrated the application of FL in training ML to detect code smells in different companies' code bases without the need to share those bases. Further, we presented an evolved code review life cycle that integrates our approach. Furthermore, we introduced a variety of datasets that targeted different organizations with code smells, and the outcome showed a higher accuracy not only in the evaluation metrics but also in the global model across all organisations.

The FedCSD global model outperforms the cross-evaluation models, where the FedCSD model was able to detect more smells on a global level which is not detectable individually by the centralized model. Moreover, The FedCSD model shows stability and robustness compared to the results of experiments 1 and 2; even the centralized model of the first and third datasets obtained the highest accuracy, where more resources are required, and there is no data privacy preservation has been considered.

The novelty that backs this study shows a higher relevance when exploring code smells using FL, with dataset two achieving the lowest accuracy of 92.30% with fewer smells in Experiment 1, while datasets one and three achieved the highest accuracy with a slight difference of 98.90% and 99.5%, respectively. Consequently, in Experiment 2, a significant drop in the model accuracy, lowest accuracy 63.80% is seen where fewer smells exist in the training dataset. Ultimately, in Experiment 3, where the dataset is split into 10 companies, an accuracy of 98.34% was achieved by the global model that has been trained using 10 companies for 100 training rounds. In addition, we presented relevant studies that have utilized federated learning in a closely matching context in order to consolidate the key problem and the propositions in this paper. As a result, given that varying datasets have been used, it is the authors opinion that this study outperforms the state-of-the art FL methods. Based on above-mentioned premise, the key objective of this paper, which was identified in the earlier sections, has been reported correctly to best of our knowledge.

In view of the fore-goings, the authors reiterates that privacy being a perennial challenge among organizations, these propositions gives a guarantee of not only maintaining and preserving privacy but also an assurance of software quality through a FL code smell detection approach. However, owing to the emerging diversification in this area, there are avenues for future work.

In future work, we plan to apply our proposed approach in practice by involving multiple software development companies that develop software systems in different domains. In addition, we plan to extend the approach to detect more types of code smells in software projects implemented in various programming languages. Also, it would be imperative to explore security vulnerabilities, adversarial learning in FedCSD and mitigation strategies.

REFERENCES

[1] Martin Fowler and Kent Beck. Refactoring: improving the design of existing code. Addison-Wesley Professional, 1999.
[2] Francisco Pérez. Refactoring Planning for Design Smell Correction in Object-Oriented Software. PhD thesis, School of Engineering, Valladolid University, 2011.
[3] Amjad AbuHassan, Mohammad Alshayeb, and Lahouari Ghouti. Software smell detection techniques: A systematic literature review. Journal of Software: Evolution and Process, 33(3):e2320, 2021.
[4] K. Alkharabsheh, Y. Crespo, E. Manso, and J.A. Taboada. Software Design Smell detection: a systematic mapping study. Software Quality Journal, 2018.
[5] William H Brown, Raphael C Malveau, Hays W McCormick, and Thomas J Mowbray. AntiPatterns: refactoring software, architectures, and projects in crisis. John Wiley & Sons, Inc., 1998.
[6] Ward Cunningham. The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger, 4(2):29–30, 1993.
[7] Munkhnasan Choinzon and Yoshikazu Ueda. Detecting defects in object oriented designs using design metrics. In J. Conf. on Knowledge-Based Software Engineering, pages 61–72, 2006.
[8] Rahma Fourati, Nadia Bouassida, and Hanêne Abdallah. A metric-based approach for anti-pattern detection in UML designs. Computer and Information Science, pages 17–33, 2011.
[9] Cristina Marinescu, Radu Marinescu, Petru Florin Mihancea, and R Wettel. iPlasma: An integrated platform for quality assessment of object-oriented design. In Intl. Conf. Software Maintenance - Industrial and Tool Volume, pages 77–80, 2005.
[10] Naouel Moha and Yann-Gael Guéhéneuc. DECOR: a tool for the detection of design defects. In Intl. Conf. on Automated Software Engineering, pages 527–528, 2007.
[11] Matthew James Munro. Product metrics for automatic identification of "bad smell" design problems in java source-code. In Intl. Conf. Software Metrics, pages 15–15, 2005.
[12] Raed Shatnawi. Deriving metrics thresholds using log transformation. J. Software: Evolution and Process, 27(2):95–113, 2015.
[13] K. Alkharabsheh, Y. Crespo, M. Fernandez-Delgado, J.R. Viqueira, and J.A. Taboada. Exploratory study of the impact of project domain and size category on the detection of the god class design smell. Software Quality Journal, 2021.
[14] Salima Hassaine, Foutse Khomh, Yann-Gaël Guéhéneuc, and Sylvie Hamel. Ids: an immune-inspired approach for the detection of software design smells. In Intl. Conf. Quality of Information and Communications Technology, pages 343–348, 2010.

[15] Foutse Khomh, Stephane Vaucher, Yann-Gaël Guéhéneuc, and Houari Sahraoui. BDTEX: A GQM-based Bayesian approach for the detection of antipatterns. J. Systems and Software, 84(4):559–572, 2011.

[16] Jochen Kreimer. Adaptive detection of design flaws. Electronic Notes in Theoretical Computer Science, 141(4):117–136, 2005.

[17] Nakarin Maneerat and Pomsiri Muenchaisri. Bad-smell prediction from software design model using machine learning techniques. In Intl. J. Conf. on Computer Science and Software Engineering, pages 331–336, 2011.

[18] Fabiano Pecorelli, Dario Di Nucci, Coen De Roover, and Andrea De Lucia. A large empirical assessment of the role of data balancing in machine-learning-based code smell detection. Journal of Systems and Software, 169:110693, 2020.

[19] F. Pecorelli, F. Palomba, D. Di Nucci, and A. De Lucia. Comparing heuristic and machine learning approaches for metric-based code smell detection. In 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pages 93–104, 2019.

[20] Ahmed Al-Shaaby, Hamoud Aljamaan, and Mohammad Alshayeb. Bad smell detection using machine learning techniques: A systematic literature review. Arabian Journal for Science and Engineering, 45, 01 2020.

[21] Muhammad Ilyas Azeem, Fabio Palomba, Lin Shi, and Qing Whang. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Information and Software Technology, 108:115–138, 4 2019.

[22] Francesca Arcelli Fontana, Mika V Mäntylä, Marco Zanoni, and Alessandro Marino. Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, 21(3):1143–1191, 2016.

[23] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. Machine learning from theory to algorithms: An overview. Journal of Physics: Conference Series, 1142:012012, 11 2018.

[24] Rajwant Singh Rao, Seema Dewangan, Alok Mishra, and Manjari Gupta. A study of dealing class imbalance problem with machine learning methods for code smell severity detection using pca-based feature selection technique. Scientific Reports, 13(1):16245, 2023.

[25] Seema Dewangan, Rajwant Singh Rao, Alok Mishra, and Manjari Gupta. Code smell detection using ensemble machine learning algorithms. Applied Sciences, 12(20):10321, 2022.

[26] Seema Dewangan, Rajwant Singh Rao, Alok Mishra, and Manjari Gupta. A novel approach for code smell detection: an empirical study. IEEE Access, 9:162869–162883, 2021.

[27] Feras M Awaysheh, Riccardo Tommasini, and Ahmed Awad. Big data analytics from the rich cloud to the frugal edge. In 2023 IEEE International Conference on Edge Computing and Communications (EDGE), pages 319–329. IEEE, 2023.

[28] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine, 37(3):50–60, 2020.

[29] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. Knowledge-Based Systems, 216:106775, 2021.

[30] Ghulam Rasool and Zeeshan Arshad. A review of code smell mining techniques. J. Software: Evolution and Process, 27(11):867–895, 2015.

[31] Tomasz Lewowski and Lech Madeyski. How far are we from reproducible research on code smell detection? a systematic literature review. Information and Software Technology, 144:106783, 2022.

[32] Michele Lanza and Radu Marinescu. Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media, 2007.

[33] Khalid Alkharabsheh. Improving design smell detection for adoption in industry. PhD thesis, CITIUS, Universidade de Santiago de Compostela, 2019.

[34] Khalid Alkharabsheh. An empirical study on the co-occurrence of design smells in the same software module:god class case study. In 2021 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), pages 1–6, 2021.

[35] Khalid Alkharabsheh, Sadi Alawadi, Yania Crespo, M. Esperanza Manso, and José A. Taboada González. Analysing agreement among different evaluators in god class and feature envy detection. IEEE Access, 9:145191–145211, 2021.

[36] Khalid Alkharabsheh, Sadi Alawadi, Karam Ignaim, Nabeel Zanoon, Yania Crespo, Esperanza Manso, and José A. Taboada. Prioritization of god class design smell: A multi-criteria based approach. Journal of King Saud University - Computer and Information Sciences, 2022.

[37] Khalid Alkharabsheh, Sadi Alawadi, Victor R Kebande, Yania Crespo, Manuel Fernández-Delgado, and José A Taboada. A comparison of machine learning algorithms on design smell detection using balanced and imbalanced dataset: A study of god class. Information and Software Technology, 143:106736, 2022.

[38] Khalid Alkharabsheh, Yania Crespo, Esperanza Manso, and J Taboada. Comparación de herramientas de detección de design smells. In Jornadas de Ingeniería del Software y Bases de Datos, pages 159–172, 2016.

[39] Khalid Alkharabsheh, Yania Crespo, Esperanza Manso, and J Taboada. Sobre el grado de acuerdo entre evaluadores en la detección de design smells. In Jornadas de Ingeniería del Software y Bases de Datos, pages 143–157, 2016.

[40] Steve Counsell and Emilia Mendes. Size and frequency of class change from a refactoring perspective. In Int. Conf. on Software Evolvability, pages 23–28, 2007.

[41] Aiko Yamashita, Marco Zanoni, Francesca Arcelli Fontana, and Bartosz Walter. Inter-smell relations in industrial and open source systems: A replication and comparative analysis. In Intl. Conf. on Software Maintenance and Evolution, pages 121–130, 2015.

[42] Abdou Maiga, Nasir Ali, Neelesh Bhattacharya, Aminata Sabané, Yann-Gaël Guéhéneuc, Giuliano Antoniol, and Esma Aïmeur. Support vector machines for anti-pattern detection. In Intl. Conf. Automated Software Engineering, pages 278–281, 2012.

[43] Manjula Peiris and James H Hill. Towards detecting software performance anti-patterns using classification techniques. ACM SIGSOFT Software Engineering Notes, 39(1):1–4, 2014.

[44] Mohamed Ragab, Riccardo Tommasini, Feras M Awaysheh, and Juan Carlos Ramos. An in-depth investigation of large-scale rdf relational schema optimizations using spark-sql. 2021.

[45] Feras M Awaysheh, Mamoun Alazab, Sahil Garg, Dusit Niyato, and Christos Verikoukis. Big data resource management & networks: Taxonomy, survey, and future directions. IEEE Communications Surveys & Tutorials, 2021.

[46] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine, 37(3):50–60, 2020.

[47] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. IEEE Communications Surveys & Tutorials, 2020.

[48] Zhipeng Cai and Zaobo He. Trading private range counting over big iot data. In 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pages 144–153. IEEE, 2019.

[49] Junjie Pang, Yan Huang, Zhenzhen Xie, Qilong Han, and Zhipeng Cai. Realizing the heterogeneity: A self-organized federated learning framework for iot. IEEE Internet of Things Journal, 2020.

[50] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. arXiv preprint arXiv:1902.01046, 2019.

[51] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics, pages 1273–1282. PMLR, 2017.

[52] Feras M Awaysheh, Mohammad N Aladwan, Mamoun Alazab, Sadi Alawadi, José C Cabaleiro, and Tomás F Pena. Security by design for big data frameworks over cloud computing. IEEE Transactions on Engineering Management, 2021.

[53] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In Proceedings of the 12th ACM workshop on artificial intelligence and security, pages 1–11, 2019.

[54] Ronald Doku, Danda B Rawat, and Chunmei Liu. Towards federated learning approach to determine data relevance in big data. In 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI), pages 184–192. IEEE, 2019.

[55] Sadi Alawadi, Victor R Kebande, Yuji Dong, Joseph Bugeja, Jan A Persson, and Carl Magnus Olsson. A federated interactive learning iot-based health monitoring platform. In European Conference on Advances in Databases and Information Systems, pages 235–246. Springer, 2021.

[56] Artrim Kjamilji, Erkay Savaş, and Albert Levi. Efficient secure building blocks with application to privacy preserving machine learning algorithms. IEEE Access, 9:8324–8353, 2021.

[57] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014.
[58] Victor R Kebande, Sadi Alawadi, Feras M Awaysheh, and Jan A Persson. Active machine learning adversarial attack detection in the user feedback process. IEEE Access, 9:36908–36923, 2021.
[59] Victor R. Kebande, Sadi Alawadi, Joseph Bugeja, Jan A. Persson, and Carl Magnus Olsson. Leveraging federated learning & blockchain to counter adversarial attacks in incremental learning. In 10th International Conference on the Internet of Things Companion, pages 1–5, 2020.
[60] Marc Haller, Christian Lenz, Robin Nachtigall, Feras M Awayshehl, and Sadi Alawadi. Handling non-iid data in federated learning: An experimental evaluation towards unified metrics. In 2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), pages 0762–0770. IEEE, 2023.
[61] Paul Johannesson and Erik Perjons. An introduction to design science, volume 10. Springer, 2014.
[62] Runeson Per and Höst Martin. Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Engg., 14(2):131–164, 2009.
[63] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. Experimentation in Software Engineering. Springer, 2012.
[64] Feras M Awaysheh. From the cloud to the edge towards a distributed and light weight secure big data pipelines for iot applications. In Trust, Security and Privacy for Big Data, pages 50–68. CRC Press, 2022.
[65] Morgan Ekmefjord, Addi Ait-Mlouk, Sadi Alawadi, Mattias Åkesson, Desislava Stoyanova, Ola Spjuth, Salman Toor, and Andreas Hellander. Scalable federated machine learning with fedn. arXiv preprint arXiv:2103.00148, 2021.
[66] Khalid Alkharabsheh, Yania Crespo, Manuel Fernández Delgado, José Manuel Cotos, and José A. Taboada. Assessing the influence of size category of the project in god class detection, an experimental approach based on machine learning. In Angelo Perkusich, editor, The 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, Hotel Tivoli, Lisbon, Portugal, July 10-12, 2019, pages 361–472. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2019.
[67] Francesca Arcelli Fontana, Mika V Mäntylä, Marco Zanoni, and Alessandro Marino. Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, 21(3):1143–1191, 2016.
[68] k. Alkharabsheh, S. Alawadi, V. Kebande, Y. Crespo, M. Delgado, and J. Taboada. Replication package of raw data, scripts and all necessary material for replication, 2021.
[69] Bryan Alfason Sunjaya, Syarifah Diana Permai, and Alexander Agung Santoso Gunawan. Forecasting of covid-19 positive cases in indonesia using long short-term memory (lstm). Procedia Computer Science, 216:177–185, 2023.
[70] Salman Toor, Mathias Lindberg, Ingemar Falman, Andreas Vallin, Olof Mohill, Pontus Freyhult, Linus Nilsson, Martin Agback, Lars Viklund, Henric Zazzik, Ola Spjuth, Marco Capuccini, Joakim Möller, Donal Murtagh, and Andreas Hellander. Snic science cloud (ssc): A national-scale cloud infrastructure for swedish academia. In 2017 IEEE 13th International Conference on e-Science (e-Science), pages 219–227, 2017.
[71] Nader Bouacida and Prasant Mohapatra. Vulnerabilities in federated learning. IEEE Access, 9:63229–63249, 2021.
[72] Xingchen Zhou, Ming Xu, Yiming Wu, and Ning Zheng. Deep model poisoning attack on federated learning. Future Internet, 13(3):73, 2021.
[73] Florian Nuding and Rudolf Mayer. Data poisoning in sequential and parallel federated learning. In Proceedings of the 2022 ACM on International Workshop on Security and Privacy Analytics, pages 24–34, 2022.
[74] Haotian Liang, Youqi Li, Chuan Zhang, Ximeng Liu, and Liehuang Zhu. Egia: An external gradient inversion attack in federated learning. IEEE Transactions on Information Forensics and Security, 2023.
[75] Anshuman Suri, Pallika Kanani, Virendra J Marathe, and Daniel W Peterson. Subject membership inference attacks in federated learning. arXiv preprint arXiv:2206.03317, 2022.
[76] Alexander Galozy, Sadi Alawadi, Victor Kebande, and Sławomir Nowaczyk. Beyond random noise: Insights on anonymization strategies from a latent bandit study. arXiv preprint arXiv:2310.00221, 2023.

SADI ALAWADI received his Ph.D. in Computer Science/AI from the Research Center of Intelligent Technologies (CiTIUS) at the University of Santiago de Compostela, Spain, in 2018. He also holds a Master's degree in Soft Computing and Intelligent Systems from Granada University, awarded in 2012. Currently, he works as an associate professor at the Blekinge Institute of Technology, Sweden. His academic journey has seen him in various research and teaching roles, including his previous position as an Assistant Professor at Halmstad University, Sweden. He has also held Postdoctoral Research positions at esteemed institutions such as the Department of Information Technology, Division of Scientific Computing at Uppsala University, the IOTAP Research Center at Malmo¨ University, and the Consiglio Nazionale delle Ricerche (CNR) - ISTI in Pisa, Italy. He has several publications in top-tier journals and conferences, including Neural Networks, IEEE Transactions on Engineering Management (TEM), IEEE Transactions on Industrial Informatics (TII), INFSOF, NCA, JKSUCIS, CCGRID, and more. His primary research interests encompass a wide range of cutting-edge areas, such as the Internet of Things (IoT), Machine Learning (ML), Deep Learning, Real-time Analysis, Data Visualization, Big Data, Digital Forensics, Edge and Cloud Computing, Dimensionality Reduction, Blockchain, Federated Learning, Transfer and Interactive Learning, and Internet of Health.

KHALID ALKHARABSHEH received a B.Sc. 2002 and an M.Sc. 2005 degree in computer science from Yarmouk University and Al-Balqa Applied University, Jordan, respectively. He was appointed as a lecturer at Al-Balqa Applied University (BAU) in September 2006, won an Erasmus Mundus grant to pursue his Ph.D. in 2014 from the Research Center of Intelligent Technologies (CiTIUS) at Santiago de Compostela University in Spain, and was awarded his Ph.D. in 2019. Dr. Alkharabsheh was appointed as an assistant professor in the software engineering department in 2019 and the head of the department from 2021 to date. Alkharabsheh's research interests include machine learning, big data, software quality, empirical software engineering, software validation and verification, and design smell detection. He is currently an assistant professor and works with different research teams and committees.

FAHED ALKHABBAS is an Assistant Professor at the department of computer science and media technology at Malmö University, Sweden. He is also affiliated with the Internet of Things and People (IoTaP) Research Centre, Sweden. Fahed received his Ph.D. in computer science from Malmö University. He also holds a master's degree in computer science from Trento University, Italy, and a bachelor's degree in computer information technology from the Arab American University, Palestine. His research focus is on the overlap between software engineering, artificial intelligence, and internet of things. Dr. Fahed has more than seven years of industrial experience in software development, during which he lead several teams in international organizations and companies, such as the United Nations, Tetra Tech DPK, GVC-Italia, and FreightOS.

VICTOR R KEBANDE , Member, IEEE, received aa Ph.D. in computer science (information and computer security architectures and digital forensics) from the University of Pretoria, Hatfield, South Africa. He was a Researcher with the Information and Computer Security Architectures (ICSA) and the DIgiFORS Research Groups, University of Pretoria, and he was a Postdoctoral Researcher with the Internet of Things and People (IOTAP) Center, Department of Computer Science, Malmö University, Malmö, Sweden. He was also a Postdoctoral Researcher of cyber and information security in information systems research subject with the Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden. He is currently an Assistant Professor of IT Security with the Department of Computer Science (DIDA), Blekinge Institute of Technology (BTH), Karlskrona, Sweden. His research interests include cyber, information security, and digital forensics in the IoT, IoT security, digital forensics-incident response, cyber-physical system protection, critical infrastructure protection, cloud computing security, computer systems, distributed system security, threat hunting and modeling, cyber-security risk assessment, Blockchain technologies, and privacy-preserving techniques. He is also an Editorial Board Member for Forensic Science International: Reports journal.

FERAS M. AWAYSHEH Hold Ph.D.in Big Data and Cloud Computing from the University of Santiago de Compostela in Spain, following his BSc. in Software Engineering from Al Balqa' Applied University and an Honors MSc. from the New York Institute of Technology (NYIT), where he specialized in Information, Computer, and Network Security. Currently, he is leading the Edge Intelligence and Data Analytics Research Group at Tartu University. His previous roles have enriched his international experience, including a visiting fellow at the University of Edinburgh, UK, and an adjunct lecturer position at Charles Darwin University, Australia. Dr. Awaysheh's service as a Springer's Cluster Computing journal and IEEE Transaction Service Computing associate editor and Guest editor at Elsevier Information Processing and Management Paradigms and Future Generation Computer Systems. He has several publications in top-tier journals and conferences, including IEEE Communications Surveys & Tutorials, IEEE Transactions (II, EM, and IoT) and FGCS. He has led as the general chair and technical program chair for several conferences, including FMEC 2023, IDSTA2021, IEEE EDGE, IEEE CBDCom, iMETA, FLTA, and MegaData.

FABIO PALOMBA is an Assistant Professor at the Software Engineering (SeSa) Lab of the University of Salerno. He received the European PhD degree in Management & Information Technology in 2017. His PhD Thesis was the recipient of the 2017 IEEE Computer Society Best PhD Thesis Award. His research interests include software maintenance and evolution, software engineering for artificial intelligence, empirical software engineering, source code quality, and mining software repositories. He was the recipient of multiple awards and research grants for his research and, in 2023, he was awarded with the prestigious IEEE/TCSE Rising Star Award. He has been and is currently involved in the Editorial Board of several software engineering journals and in Program Committee of multiple software engineering conferences.

MOHAMMED AWAD a Professor of Computer Engineering/Artificial intelligence. He received a B.S. Degree in Automation Engineering from Palestine Polytechnic University in the year 2000, he studied for a master's & Ph.D. in Computer Engineering at Granada University Spain in the Artificial intelligence field. From 2005 to 2006, he was a contract Researcher at Granada University in the research group Computer Engineering: Perspectives and Applications. Since Feb. 2006, he has been an Assistant Professor in the Computer Engineering Department, College of Engineering and Information Technology at Arab American University, Palestine. In 2016 he was promoted to the rank of Full Professor in Computer Engineering. Prof. Awad worked for more than 18 years at the Arab American University in an academic Position, in parallel with various Academic administrative positions. Through his research and educational experience, Prof. Awad has developed a strong research record. he concentrates on using AI tools in real applications, especially in using AI in the health sector. His research interests include Artificial Intelligence Techniques, Machine Learning, Deep Learning, Neural Networks, Function Approximation of Complex Systems, Clustering Techniques, Optimization Algorithms, and Time-series Prediction. He won several awards and research grants.