Federated Repair of Deep Neural Networks

Davide Li Calsi
Technical University of Munich
Munich, Germany
davide.li-calsi@tum.de

Thomas Laurent
National Institute of Informatics, JSPS International Research Fellow
Tokyo, Japan
thomas-laurent@nii.ac.jp

Paolo Arcaini
National Institute of Informatics
Tokyo, Japan
arcaini@nii.ac.jp

Fuyuki Ishikawa
National Institute of Informatics
Tokyo, Japan
f-ishikawa@nii.ac.jp

ABSTRACT

As DNNs are embedded in more and more critical systems, it is essential to ensure that they perform well on specific inputs. DNN repair has shown good results in fixing specific misclassifications in already trained models using additional data, even surpassing additional training. In safety-critical applications, such as autonomous driving, collaboration between industrial actors would lead to more representative datasets for repair, that would enable to obtain more robust models and thus safer systems. However, these companies are reluctant to share their data, to both protect their intellectual property and the privacy of their users. Federated Learning is an approach that allows for collaborative, privacy-preserving training of DNNs. Inspired by this technique, this work proposes *Federated Repair* in order to collaboratively repair a DNN model without the need for sharing any raw data. We implemented Federated Repair based on a state-of-the-art DNN repair technique, and applied it to three DNN models, with federation size from 2 to 10. Results show that Federated Repair can achieve the same repair efficiency as non-federated DNN repair using the pooled data, despite the presence of rounding errors when aggregating clients' results.

CCS CONCEPTS

• **Software and its engineering** → **Search-based software engineering**; **Software testing and debugging**.

KEYWORDS

Deep Neural Networks, DNN repair, federation

ACM Reference Format:
Davide Li Calsi, Thomas Laurent, Paolo Arcaini, and Fuyuki Ishikawa. 2024. Federated Repair of Deep Neural Networks. In *2024 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest '24), April 20, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

DeepTest '24, April 20, 2024, Lisbon, Portugal

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep Neural Networks (DNNs) are at the heart of more and more critical, human-centric processes. They can be found in systems from medical diagnosis to autonomous driving, with attempts of integrating them even in judicial processes. Faults of these models can thus have dramatic consequences, even fatal ones. As such, being able to assess and rely on the quality of those models and the system they power is essential. The software engineering community has invested significant effort in this task [16, 19, 32], leveraging trusted methods developed for classic software systems.

DNN repair is a technique that, given a set of inputs exposing a fault in a DNN, aims at fixing it while minimally disrupting the network's other behaviours, i.e., avoiding regressions. DNN repair is particularly relevant in critical applications such as autonomous driving, where not all faulty behaviours have equal impact (e.g., mishandling inputs related to pedestrians is more severe), and thus overall accuracy is not the main goal. Repair techniques targeting DNN weights [12, 13, 15, 23–25] have been applied in this domain. They rely on two mechanisms to repair the network: (i) *Fault Localisation* (FL) to identify suspicious weights that are causing the faulty behaviour, (ii) *weight optimisation* to repair the suspicious weights, computing their new values by search.

The effectiveness of these techniques depends on the amount and quality of the data used to repair the DNN [4]. To tackle this issue, collaboration between different parties, in the form of data sharing, would lead to more diverse and representative data and thus to better repair results. However, despite the potential benefits to model quality, our collaborations with the automotive domain have shown that companies and users alike are reluctant to share data, both for IP protection and privacy reasons.

Similar opportunities and reluctance exist when training DNNs, and have lead to a family of techniques known as *Federated Learning* [30]. These techniques enable collaborative training of DNNs, pooling data from different sources without the need to actually share raw data. The benefits of the larger dataset can then be reaped without breaching the different parties' privacy.

Taking inspiration from federated learning, we propose *Federated Repair* (FedRep) to collaboratively repair DNNs as a federation, without the need for sharing data among federation's members.

Performing DNN repair as a federation is not trivial, as both the fault localisation and optimisation steps must be distributed over a now fragmented (among the members of the federation) dataset.

Moreover, in order to perform an effective repair, the approach must still share some *intermediate computation results*; it is important that the original members' raw data cannot be retrieved starting from these intermediate results. To tackle these issues, this work introduces mechanisms to split the computation of the metrics needed for DNN repair (e.g., suspiciousness scores or fitness values) over different datasets and re-aggregate them, and incorporates cryptography techniques to ensure data privacy.

We implemented FedRep based on Arachne, a state-of-the-art DNN repair technique, and applied it to three DNN models trained and repaired using a well known driving images classification dataset, with federation size from 2 to 10. Comparison with Arachne in this setting shows that although FedRep produces slightly different results, it mostly identifies the same faulty weights as Arachne and produces repaired models of the same quality. Moreover, the distribution of data amongst clients can affect the performance of the approach.

Paper structure. Sect. 2 introduces necessary preliminaries, and Sect. 3 presents the proposed approach FedRep. Sect. 4 describes the experiment design, and Sect. 5 discusses experimental results. Finally, Sect. 6 reviews threats to the validity of this work, Sect. 7 discusses related work, and Sect. 8 concludes the paper.

2 PRELIMINARIES

Our proposal for federated repair is based on the state-of-the-art classification DNN repair technique Arachne [23]. This method splits the repair process into two subtasks:

Fault Localisation (FL): aims at finding a set of suspicious weights, i.e., a set of weights that are likely to cause a target fault (one or more misclassifications);

Weights optimisation: modifies the values of the suspicious weights using evolutionary computation to fix the fault without introducing regressions.

Arachne uses a dataset D^R to repair a DNN $\mathcal{M}$. D^R is split into:

- A *negative dataset* D^R_{neg}: inputs $\mathcal{M}$ misclassifies;
- A *positive dataset* D^R_{pos}: inputs $\mathcal{M}$ correctly classifies.

Fault localisation. It uses two metrics to understand the role of each weight w. As reported in [1], the *gradient loss* is:

$$G_{loss}(D^R, w) = \frac{1}{|D^R|} \sum_{x \in D^R} \frac{\partial \mathcal{L}(w, x)}{\partial w} \tag{1}$$

with $\mathcal{L}$ the loss function. As reported in [1], the *forward impact* is:

$$F_{imp}(D^R, w) = A(D^R, w) \cdot G_{out}(D^R, w) \tag{2}$$

with

$$A(D^R, w) = \frac{1}{|D^R|} \sum_{x \in D^R} \frac{w \cdot o(w)[x]}{\sum_{w' \in L(w)} w' \cdot o(w')[x]} \tag{3}$$

$$G_{out}(D^R, w) = \frac{1}{|D^R|} \sum_{x \in D^R} \frac{\partial \mathcal{M}(x)}{\partial o'(w)} \tag{4}$$

The forward impact of a weight on one dataset is the product of two scores: (i) $A(D^R, w)$ is the normalised *average activation*, where $o(w)$ is the activation function of the predecessor neuron of w, and $o(w)[x]$ is the value of this function evaluated on input x. This value is multiplied by w and normalised. The normalisation consists in dividing by the sum of the quantities $w' \cdot o(w')[x]$ for every weight w' in the same layer as w ($L(w)$ in the above equation). (ii) $G_{out}(D^R, w)$ is the average gradient to the model's output, where $o'(w)$ is the activation of the successor of w.

G_{loss} and F_{imp} are computed for each weight in the weights $\mathcal{M}.W$, and aggregated to compute two suspiciousness scores:

$$Sus_{GL}(w) = \frac{G_{loss}(D^R_{neg}, w)}{1 + G_{loss}(D^R_{pos}, w)} \tag{5}$$

$$Sus_{FI}(w) = \frac{F_{imp}(D^R_{neg}, w)}{1 + F_{imp}(D^R_{pos}, w)} \tag{6}$$

These scores reflect how important each weight w was when $\mathcal{M}$ made a wrong classification compared to its importance when $\mathcal{M}$'s classifications were correct. The FL process returns as *suspicious weights susW* the weights that form a Pareto front in the space of these two suspiciousness scores.

Weights optimisation. Arachne's optimisation targets suspicious weights *susW*, optimising their values using Differential Evolution, maximising the following fitness function fit_α:

$$fit_\alpha(D^R, cSol) = \sum_{x \in D^R_{pos}} score(x, cSol) + \alpha \sum_{x \in D^R_{neg}} score(x, cSol)$$
$$\text{with } score(x, cSol) = \begin{cases} 1 & miscl(x, cSol) = false \\ \frac{1}{\mathcal{L}(x)+1} & miscl(x, cSol) = true \end{cases} \tag{7}$$

where *cSol* is a candidate solution, and α a real-valued hyperparameter to balance the impact of positive and negative inputs, balancing between preventing regressions and fixing the fault. The score of *cSol* on a single input $score(x, cSol)$ reflects how well the candidate solution classified x, with a score of 1 if it classified it correctly, and the inverse of the loss on x otherwise.

3 FedRep – FEDERATED REPAIR OF DNNs

This section presents the proposed federated repair approach. Sect. 3.1 describes the federation, its actors, and the steps they must follow through different algorithms. Sect. 3.2 proves the theoretical correctness of these algorithms and that they lead to the same repair as obtained in the classic, single-user process. Finally, Sect. 3.3 discusses the introduction of cryptography to further protect privacy, and Sect. 3.4 estimates the overhead introduced by federation.

3.1 Context and approach description

FedRep is designed for the following scenario: n actors (e.g., companies, or clients of a company) are using the same classification model $\mathcal{M}$ and want to cooperatively repair some fault(s) (misclassification type(s)) in $\mathcal{M}$ without sharing their data (e.g., images taken while driving and manually labelled). In order to minimise the communication necessary for this cooperation to happen, FedRep adopts a client-server architecture, where each actor represents a client $c_i \in C$, and only communicates with a central server S.

As each client $c_i \in C$ does not want to disclose its dataset D^R_i, the server can not perform the computations necessary to identify and optimise suspicious weights as detailed in Sect. 2. Hence, FedRep defines a way for each client c_i to perform intermediate computations on D^R_i and to only communicate these intermediate results to S that then aggregates all clients' results in order to define

Algorithm 1: Federated FL – Client c_i

Data: D_i^R: client's dataset $\mathcal{M}$: DNN to repair
Result: $susW_{\text{FedRep}}$: suspicious weights
1 **Function** FedLocaliseClient(D_i^R,$\mathcal{M}$):
2 $GL_{neg}, Act_{neg}, GO_{neg} \leftarrow \{\}$
3 $GL_{pos}, Act_{pos}, GO_{pos} \leftarrow \{\}$
4 **for** $w \in \mathcal{M}.W$ **do**
5 $GL_{neg}[w] \leftarrow G_{loss}(D^R_{neg_i}, w) \cdot |D^R_{neg_i}|$
6 $Act_{neg}[w] \leftarrow A(D^R_{neg_i}, w)) \cdot |D^R_{neg_i}|$
7 $GO_{neg}[w] \leftarrow G_{out}(D^R_{neg_i}, w) \cdot |D^R_{neg_i}|$
8 $GL_{pos}[w] \leftarrow G_{loss}(D^R_{pos_i}, w) \cdot |D^R_{pos_i}|$
9 $Act_{pos}[w] \leftarrow A(D^R_{pos_i}, w) \cdot |D^R_{pos_i}|$
10 $GO_{pos}[w] \leftarrow G_{out}(D^R_{pos_i}, w) \cdot |D^R_{pos_i}|$
11 $ServerSend(GL_{neg}, Act_{neg}, GO_{neg}, |D^R_{neg_i}|)$
12 $ServerSend(GL_{pos}, Act_{pos}, GO_{pos}, |D^R_{pos_i}|)$
13 $susW_{\text{FedRep}} \leftarrow ServerGet()$
14 **return** $susW_{\text{FedRep}}$

Algorithm 2: Federated FL – Server S

Data: $C = \{c_i\}_{1...n}$: clients, $\mathcal{M}$: DNN to repair
Result: $susW_{\text{FedRep}}$: suspicious weights
1 **Function** FedLocaliseServer(C, $\mathcal{M}$):
2 $GL_{neg}, Act_{neg}, GO_{neg}, Sizes_{neg} \leftarrow \{\}$
3 $GL_{pos}, Act_{pos}, GO_{pos}, Sizes_{pos} \leftarrow \{\}$
4 **for** $c_i \in C$ **do**
5 $GL_{neg}[c_i], Act_{neg}[c_i], GO_{neg}[c_i], Sizes_{neg}[c_i] \leftarrow ClientGet(c_i)$
6 $GL_{pos}[c_i], Act_{pos}[c_i], GO_{pos}[c_i], Sizes_{pos}[c_i] \leftarrow ClientGet(c_i)$
7 $Sus_{FI}, Sus_{GL} \leftarrow \{\}$
8 $|D^R| \leftarrow \sum_{c_i \in C} (Sizes_{neg}[c_i] + Sizes_{pos}[c_i])$
9 **for** $w \in \mathcal{M}.W$ **do**
10 $AggrGL_{neg} \leftarrow \frac{1}{|D^R|} \cdot \sum_{c_i \in C} GL_{neg}[c_i][w]$
11 $AggrAct_{neg} \leftarrow \frac{1}{|D^R|} \cdot \sum_{c_i \in C} Act_{neg}[c_i][w]$
12 $AggrGO_{neg} \leftarrow \frac{1}{|D^R|} \cdot \sum_{c_i \in C} GO_{neg}[c_i][w]$
13 //Same aggregation for GL_{pos}, Act_{pos}, and GO_{pos}
14 $Sus_{GL}[w] \leftarrow \frac{AggrGL_{neg}}{1+AggrGL_{pos}}$
15 $Sus_{FI}[w] \leftarrow \frac{AggrAct_{neg} \cdot AggrGO_{neg}}{1+AggrAct_{pos} \cdot AggrGO_{pos}}$
16 $susW_{\text{FedRep}} \leftarrow ExtractPareto(Sus_{FI}, Sus_{GL})$
17 **for** $c_i \in C$ **do**
18 $ClientSend(susW_{\text{FedRep}}, c_i)$
19 **return** $susW_{\text{FedRep}}$

the final results and communicate them back to the clients. This is done both for the FL phase and the optimisation phase, that are described in the following.

Federated fault localisation. Alg. 1 describes the Federated FL steps for the clients. Each client computes the gradient loss (G_{loss}), average activation (A), and gradient to the output (G_{out}) for each weight w in model $\mathcal{M}$ on both its dataset of negative inputs $D^R_{neg_i}$, and of positive inputs $D^R_{pos_i}$ (lines 5-10); note that each of these scores is multiplied by the size of the dataset it was computed on in order to undo the averaging seen in Eqs. 1, 3, and 4 and make these scores aggregable by the server. Once all scores are computed (stored in maps GL_{neg}, Act_{neg}, GO_{neg}, GL_{pos}, Act_{pos}, and GO_{pos}), the client sends them to the server (lines 11-12) and waits for the server to respond with the computed suspicious weights (line 13).

Alg. 2 describes the Federated FL steps for the server S. The server first receives all scores and dataset sizes from all clients (lines 4-6) and then proceeds to aggregate them. First, it computes the total dataset size $|D^R|$ (line 8), and then computes, for each weight, the average of each positive and negative score (lines 10-13). Then, it computes the two suspiciousness scores of the weight from the different clients' scores as defined in Eqs. 5 and 6 (lines 14-15). Finally, the server extracts the Pareto front of the most suspicious weights across both scores, $susW_{\text{FedRep}}$ (line 16), sends it to all clients (lines 17-18), and returns it (line 19).

Federated weights optimisation. Once the suspicious weights $susW_{\text{FedRep}}$ are computed, the server runs weights optimisation using the *Differential Evolution* (DE) algorithm, where $susW_{\text{FedRep}}$ are the search variables. However, as the server still can not access the data D^R used for repair, the fitness computation is distributed amongst the clients. Namely, at each generation of the search:

- the server sends to all the clients the current population of new values for the suspicious weights;
- each client c_i computes the fitness function $fit_\alpha(D_i^R, cSol)$ (see Eq. 7) for each individual $cSol$ on its dataset D_i^R and returns the list of fitness values for the population;
- the server sums all clients' fitness values for each individual in order to obtain its fitness on the full dataset D^R and proceeds to the next generation of DE.

When the search finishes, either because the budget (in number of generations) is reached or another stopping condition (e.g., stagnation) is reached, the server communicates the repaired model $\mathcal{M}_{fixedWeights}$ (the model with suspicious weights set to the best individual *fixedWeights* in the search) to all clients.

3.2 Correctness

This section demonstrates that using FedRep is equivalent to using Arachne locally with a dataset $D^R = \cup_{c_i \in C} D_i^R$ (i.e., FedRep's clients' datasets D_i^R form a partition of the dataset used in Arachne), and that both methods' results should be the same (see Remark 1). First it shows the mathematical equivalence of the fault localisation phase, and then of the optimisation phase.

FL correctness. In the FL phase, FedRep returns the set of suspicious weights $susW_{\text{FedRep}} = FedLoc(\mathcal{M}, D_1^R, \ldots, D_n^R)$ and Arachne returns $susW_{\text{Arachne}} = Loc(\mathcal{M}, D^R)$. We first want to show the following theorem.

Theorem 1 (Correctness of Federated FL)**.** Federated Fault Localisation is correct, i.e., for any number of clients n and any partition $D_1^R, \ldots, D_n^R$ of dataset D^R:

$$susW_{\text{FedRep}} = Loc\left(\mathcal{M}, D^R\right) = FedLoc\left(\mathcal{M}, D_1^R, \ldots, D_n^R\right) = susW_{\text{Arachne}}$$

Proof. It suffices to show that the gradient loss and forward impact computed by *Loc* are equal to those computed by *FedLoc*. Let's focus on a single weight w without loss of generality. First consider the definition of gradient loss on a dataset, i.e., $G_{loss}(D^R, w)$ in Eq. 1. Because $D_1^R, \ldots, D_n^R$ form a partition of D^R, we can write

$$G_{loss}(D^R, w) = \frac{1}{|D^R|}\left(\sum_{x \in D_1^R} \frac{\partial \mathcal{L}(w, x)}{\partial w} + \ldots + \sum_{x \in D_n^R} \frac{\partial \mathcal{L}(w, x)}{\partial w}\right)$$

We can also show that:

$$\sum_{x \in D_i^R} \frac{\partial \mathcal{L}(w, x)}{\partial w} = G_{loss}(D_i^R, w) \cdot |D_i^R|$$

allowing us to conclude:

$$G_{loss}(D^R, w) = \frac{1}{|D^R|} \sum_{i=1}^{n} G_{loss}(D_i^R, w) \cdot |D_i^R|$$

Moving on to the forward impact, it is *not* true that $F_{imp}(D^R, w) = \frac{1}{|D^R|} \sum_{i=1}^{n} F_{imp}(D_i^R, w) \cdot |D_i^R|$. However, Eq. 2 defines the forward impact on a dataset as the product of two intermediate scores, namely $A(D^R, w)$ and $G_{out}(D^R, w)$. Repeating the same reasoning that holds for gradient loss, we find that

$$A(D^R, w) = \frac{1}{|D^R|} \sum_{i=1}^{n} |D_i^R| \cdot A(D_i, w) = AggrAct$$

$$G_{out}(D^R, w) = \frac{1}{|D^R|} \sum_{i=1}^{n} |D_i| \cdot G_{out}(D_i^R, w) = AggrGO$$

Once these scores are aggregated for both D_{neg}^R and D_{pos}^R, they are combined into $Sus_{GL}(w)$ and $Sus_{FI}(w)$ (see Eqs. 5 and 6). The procedure is done for each weight w, and then Federated FL extracts the Pareto front like Arachne's FL. Since for each w the suspiciousness scores are equal to those computed by Arachne, the resulting Pareto front is the same, i.e., $susW_{\text{FedRep}} = susW_{\text{Arachne}}$. □

The previous proof reduces the computation of forward impact and gradient loss to the computation of fundamental linear scores. $G_{loss}(D^R, w)$ is already linear, needing no decomposition. On the other hand, we break $F_{imp}(D^R, w)$ down into the product of $A(D^R, w)$ and $G_{out}(D^R, w)$. By aggregating these scores independently and multiplying them on the server's side, we can federate the forward impact computation.

Weights optimisation correctness. We here prove that the weights optimisation computed by the federated DE is correct.

Theorem 2 (Correctness of Federated DE)**.** The federated fitness computation is correct, i.e, for any candidate solution *cSol*, number of clients n, dataset D^R, and partitioning $D_1^R, \ldots, D_n^R$ of D^R:

$$fit_{\alpha \text{FedRep}}(D^R, cSol) = \sum_{i=1}^{n} fit_\alpha(D_i^R, cSol) = fit_\alpha(D^R, cSol)$$

Proof. The claim follows from fit_α's linearity. Since we are considering a partition of D^R, we can rewrite Eq. 7 as

$$\begin{aligned} fit_\alpha(D^R, cSol) &= \sum_{i=1}^{n} \sum_{x \in D_{pos_i}^R} score(x, cSol) + \alpha \sum_{i=1}^{n} \sum_{x \in D_{neg_i}^R} score(x, cSol) \\ &= \sum_{i=1}^{n} \left(\sum_{x \in D_{pos_i}^R} score(x, cSol) + \alpha \sum_{x \in D_{neg_i}^R} score(x, cSol) \right) \\ &= \sum_{i=1}^{n} fit_\alpha(D_i^R, cSol) = fit_{\alpha \text{FedRep}}(D^R, cSol) \end{aligned}$$

□

Remark 1. The above proofs are only theoretical, and consider exact operations. However, floating point arithmetics, as performed by computing devices, are not exact. The approximations and errors resulting from this imprecision accumulate in different ways depending on the specific hardware and software used, and even the order of operations. Hence, although FedRep is mathematically equivalent to Arachne, it could produce different results in practice. The difference between FedRep and Arachne in practice is explored in the experiments (see Sects. 4 and 5).

3.3 Cryptography

As a countermeasure to processes such as gradient inversion attacks [10] that can extract data about the original dataset from aggregated metrics like the ones sent from clients to the server, one may add a layer of encryption to FedRep. In particular, we refer to standard techniques for secure aggregation [3]. For the sake of simplicity, we assume that no client will drop out during the repair process, although techniques exist that would allow such a setting, which could be used in practice.

At the beginning of each phase (FL or DE), each pair of clients (c_i, c_j) runs a key-exchange protocol to establish a private key $K_{i,j}$, and agree on a bit value $b_{i,j} \in \{0, 1\}$. We use the well-known Diffie-Hellman [6] key-exchange due to its simplicity and renowned security. As a result, each client c_i has $n-1$ keys $K_{(i,1)}, \ldots, K_{(i,i-1)}$, $K_{(i,i+1)}, \ldots, K_{(i,n)}$. The keys are used as seeds of $n-1$ Pseudo-Random Number Generators (PRNGs) on c_i's side. We choose the ChaCha stream cipher and use it as a PRNG, as implemented in the RandomGen Python package [2], but any other cryptographically secure PRNG would suit. When client c_i needs to send a local score V_i to the server, it does the following:

(i) Run each PRNG, obtaining a sequence of $n-1$ chunks of bytes $B_1, \ldots B_{i-1}, B_{i+1}, \ldots, B_n$.
(ii) Convert each chunk to a real number (assume B_j encodes a real number in binary), obtaining $R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n$. The j-th element R_j is associated with client c_j.
(iii) Encrypt V_i as $E_i = V_i + \sum_{j \neq i} sign(b_{i,j}, i, j) \cdot R_j$

$$\text{with } sign(b_{i,j}, i, j) = \begin{cases} 1 & (b_{i,j} = 0 \wedge i < j) \vee (b_{i,j} = 1 \wedge i > j) \\ -1 & \text{otherwise} \end{cases}$$

The server can compute the sum of plain scores by summing the encrypted scores [3], as $\sum_{i=1}^{n} E_i = \sum_{i=1}^{n} V_i$.

3.4 Protocol overhead

Federating the DNN repair process introduces some extra complexity to enable both the distribution of the process and the protection of privacy. The resulting overhead is due to the additional *communication*, as clients exchange data with the server to aggregate local scores and with each other for key agreement, and *cryptography*, as the clients and the server perform local cryptographic computation.

Communication. We start by analysing the overhead introduced in FL. Each client computes and sends three scores for each weight, and the server replies with a Pareto front of at most $|\mathcal{M}.W|$ weights. Considering n clients, all actors exchange a total of $n \cdot (|\mathcal{M}.W| + 1)$ messages. Regarding DE, at each generation, each client sends one fitness value for each candidate solution for a total complexity of $O(n \cdot PopSize)$, where *PopSize* is the population size. Because

this is repeated at most *GenMax* times (*GenMax* is the maximum number of generations), the total load of exchanged messages is $O(n \cdot GenMax \cdot PopSize)$. On top of that, at the very beginning of FedRep, every client pair exchanges keys, for a total of $\frac{n(n-1)}{2} = O(n^2)$ exchanges. Assuming a key size (in bits) of S, the total complexity is $O(S \cdot n^2)$. Finally, we remark that the protocol requires synchronous communication in the Federated FL and DE phases. The server needs all the individual local fitness values to start the aggregation and perform one step of differential evolution, hence it must wait for the slowest client.

Computation. From the computational viewpoint, the main overhead arises from score encryption. Such a burden is only on the clients' side, as the server merely sums the received scores. The total overhead is due to modular exponentiation for Diffie-Hellman key-exchange and running the PRNG. Running the key exchange requires computing two modular powers, whose complexity depends on the exponentiation strategy. We assume a square-and-multiply strategy, which computes $b^x \mod y$ in $O(\log(x) \cdot \log(y)^2)$ steps. In our case, both x and y are S-bits numbers, where S is an integer parameter determining the hardness of inverting the discrete logarithm, leading to complexity $O(S^3)$. Recalling that such an exchange must happen for all pairs of clients, the complexity for one client is $O(S^3 \cdot n^2)$. As for the PRNG, the ChaCha algorithm generates an integer in $O(1)$ (constant time), as we assume the scores to encrypt to have a fixed size in bits. Each client runs it $O(n \cdot |\mathcal{M}.W|)$ times for FL, while DE requires $O(n \cdot GenMax \cdot PopSize)$ iterations. The total is $O(n \cdot (|\mathcal{M}.W| + GenMax \cdot PopSize))$.

4 EXPERIMENT DESIGN

This section describes the experiments we conducted to assess the effectiveness and efficiency of the approach. It first describes the research questions we aim to answer, then the benchmarks we used and the approaches we compared to answer them. Finally, it describes how the experiments were run, and the metrics used to evaluate their results. All experimental code and results are available in the replication package [14].

Research questions. To evaluate FedRep's repair results and its performance, we explore the following research questions (RQs):

- **RQ1** Do FedRep and Arachne produce the same FL results? This RQ aims at evaluating the equivalence of both methods' FL *in practice* (see Remark 1). It is divided into two sub-RQs:
  - **RQ1.1** Do FedRep and Arachne produce the same suspiciousness values Sus_{GL} and Sus_{FI}?
  - **RQ1.2** Do FedRep and Arachne produce the same sets of suspicious weights?
- **RQ2** Do FedRep and Arachne produce the same repaired model? This RQ aims at evaluating the equivalence of both methods' repair *in practice*. It is divided into two sub-RQs:
  - **RQ2.1** Do FedRep and Arachne produce the same model weights?
  - **RQ2.2** Do FedRep and Arachne produce repairs of the same quality?
- **RQ3** What is the influence of different data distributions among clients on the performance of FedRep? This RQ investigates the impact of imbalanced data distributions on FedRep's repair time. It focuses on the federated weight optimisation (FedRep's bottleneck), and in particular on how the size of the local dataset of the different clients impacts the fitness computation time of each client.

Benchmarks. We selected three DNNs for image classification, namely VGG16, VGG19, and EnetB0. Each model was trained on the BDD100K dataset, a dataset widely used for image classification in the context of Autonomous Driving Systems that contains 318,878 images of various vehicles, pedestrians, and traffic signs. We split the dataset D into three subsets: one dataset D^{Tr} for training (80%), one dataset D^{Te} for testing (16%), and one dataset D^{AllR} from which to sample images for repair (4%). We trained each model architecture on D^{Tr} for five epochs. This choice is meant to under-train the neural network, and ensure that misclassifications are present, i.e., that "there is something to repair". We picked nine misclassifications types in which images of class Cl are classified as something else; for each considered class Cl, we run the repair using a repair set D^R (with $D^R \subset D^{AllR}$) that contains as negative dataset D^R_{neg} the images of class Cl in D^{AllR} that are misclassified, and as positive dataset D^R_{pos} the images in D^{AllR} that are correctly classified. In the experiments, we restrict localisation and repair to the model's last layer to limit the computational cost, as done in other works [13, 22].

Compared approaches. We compared FedRep with Arachne. For each DNN and class under repair, we ran FedRep three times, with 2, 5, and 10 clients respectively. FedRep and Arachne use the same dataset and the same seed to prevent randomness from polluting the comparison. We set the search hyperparameters for DE to $PopSize = 100$ and $GenMax = 100$.

Running of the experiments. An *experiment* consists of the execution of one approach (either FedRep with a fixed number of clients or Arachne), over one of the three DNN models with the goal of repairing the misclassifications of one of the nine considered classes. In total, there are $3 \times 4 \times 9 = 108$ experiments.

Evaluation metrics. To answer RQ1, we compare the FL results of FedRep and Arachne. Namely, for a given model $\mathcal{M}$, class under repair Cl, and number of clients, we first measure the relative error introduced by FedRep in Sus_{GL} and Sus_{FI}; this reflects the impact of the computational errors on the suspiciousness scores. Then, we compare the suspicious weights $susW_{\text{FedRep}}$ and $susW_{\text{Arachne}}$ to assess the impact of these errors on the final FL results.
To answer RQ2, we compare the repair results of FedRep and Arachne. Specifically, for a given model $\mathcal{M}$ and class under repair Cl, we compute the L2 norm between the weight vectors of each repaired model, reflecting the difference between the models produced by the two methods. We also compare the fitness values of these two models, which reflects the quality of the models.
To answer RQ3, we record the time taken to compute the fitness for each client. For FedRep, this means recording the time it takes for one client to compute the repair fitness on its local dataset.

5 EXPERIMENT RESULTS

This section describes and discusses the results of the experiments described in Sect. 4 in order to answer the research questions.

Table 1: RQ1.1 – Numerical errors in federated FL of FedRep (inaccuracy Sus_{FI}; inaccuracy Sus_{GL})

(a) VGG16

n	Cl								
	0	1	2	3	6	7	8	9	12
2	0 ; 0	2.4e-4 ; 0	0 ; 0	0 ; 0	0 ; 0	1.3e-2 ; 8e-3	0 ; 0	2.4e-4 ; 2.4e-4	0 ; 0
5	0 ; 0	2.4e-4 ; 2.4e-4	0 ; 0	0 ; 0	0 ; 0	2.4e-4 ; 2.4e-4	0 ; 0	0 ; 0	1.2e-2 ; 9.1e-3
10	0.65 ; 0.84	2.4e-4 ; 2.4e-4	0 ; 0	0 ; 0	0 ; 0	0 ; 2.4e-4	0 ; 2.4e-4	0 ; 0	0 ; 0

(b) VGG19

n	Cl								
	0	1	2	3	6	7	8	9	12
2	2.4e-4 ; 0	9.4e-5 ; 0	3.8e-5 ; 0	1.7e-4 ; 0	7.5e-5 ; 0	1.1e-4 ; 0	9.4e-5 ; 0	7.5e-5 ; 0	7.5e-5 ; 0
5	2.6e-4 ; 0	1.5e-4 ; 0	3.8e-5 ; 0	9.4e-5 ; 0	5.6e-5 ; 0	7.5e-5 ; 0	7.5e-5 ; 0	1.7e-4 ; 0	1.7e-4 ; 0
10	2.3e-4 ; 0	1.9e-5 ; 0	5.6e-5 ; 0	1.7e-4 ; 0	5.6e-5 ; 0	2.1e-4 ; 0	2.6e-4 ; 0	1.3e-4 ; 0	9.4e-4 ; 0

(c) EnetB0

n	Cl								
	0	1	2	3	6	7	8	9	12
2	0 ; 0	0 ; 0	0 ; 0	7.57e-3 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0
5	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0
10	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0

RQ1: FL comparison. Table 1 reports results related to RQ1.1: the numerical results of the comparison of suspiciousness scores for all the repaired classes under different numbers of users, for VGG16, VGG19, and EnetB0. Each pair $(f;g)$ reports the proportion of inaccurate scores for Sus_{FI} and Sus_{GL} respectively. A federated score s_{FedRep} (either Sus_{FI} or Sus_{GL}) is deemed "inaccurate" if $|\frac{s_{\text{FedRep}}}{s_{\text{Arachne}}} - 1| > 0.01$, where s_{Arachne} is the corresponding non-federated score. The results confirm the presence of numerical errors, however relatively small.

However, when we compare $susW_{\text{Arachne}}$ and $susW_{\text{FedRep}}$ (RQ1.2), both methods produce the same suspicious weights in most cases. Repair of classes 2 and 3 with model EnetB0 are the only exceptions, in which FedRep found different suspicious weights for all numbers of clients. However, even in those cases, the difference is minimal, and FedRep only adds one weight compared to $susW_{\text{Arachne}}$. This empirically confirms the correctness of federated FL.

Answer to RQ1.1: Experimental results confirm that floating point calculations introduce errors in the suspicioussness scores produced by FedRep, although they are relatively small.
Answer to RQ1.2: The suspicious weights found by FedRep correspond to those of Arachne in most cases, despite the difference in suspicioussness scores. When a difference occurs it is minimal, and FedRep still finds all suspicious weights identified by Arachne.

RQ2: Repair comparison. To answer RQ2, we compare the repaired models obtained with Arachne with those obtained with FedRep. Table 2 shows the difference between repaired models produced by both methods for each architecture. Lines marked "L2" relate to RQ2.1 and report the L2 distance between the federated and non-federated repaired models. More precisely, for each DNN $\mathcal{M}$, each repaired class Cl, and each number of clients n, we arrange

Table 2: RQ2 – Comparison between federated (FedRep) and non-federated (Arachne) repair

(a) VGG16

n		Cl								
		0	1	2	3	6	7	8	9	12
2	L2	0.32	1.4e-4	0	0	0	0	0.93	1.13e-11	0
	fit	4.14e-4	3.07e-7	1.84e-9	5.53e-9	1.71e-9	2.19e-9	1.60e-4	3.87e-4	4.63e-9
5	L2	0	0	2.7e-6	5e-6	0	0	0	0	
	fit	2.34e-9	3.09e-7	3.23e-9	1.59e-8	2.20e-9	1.01e-9	2.19e-9	6.99e-10	4.95e-9
10	L2	0.91	0	2.3e-6	9e-7	0	0	0	0.45	0
	fit	6.00-e-3	1.67e-9	2.04e-9	3.49e-9	1.21e-9	2.01e-9	1.20e-9	3.87e-4	2.45e-9

(b) VGG19

n		Cl								
		0	1	2	3	6	7	8	9	12
2	L2	0	0	0	0	0	0	0	0	0
	fit	6.07e-5	8.58e-4	3.7e-9	4.83e-9	3.93e-8	2.17e-9	8.81e-11	7.98e-8	5.58e-9
5	L2	0.14	0.25	0	0	0	0	0.43	3.42e-5	0
	fit	1.18e-4	8.58e-4	1.32e-8	1.28e-8	5.68e-9	2.88e-10	5.47e-4	1.01e-8	1.43e-8
10	L2	0	0.23	0	0	0	0.07	0	0	0
	fit	1.08e-8	1.11e-5	7.99e-9	1.07e-8	1.20e-8	2.60e-4	1.40e-8	4.13e-9	1.64e-8

(c) EnetB0

n		Cl								
		0	1	2	3	6	7	8	9	12
2	L2	9e-16	9.4e-3	2.69	0.29	0	0	3.6e-7	0	6.9e-2
	fit	1.57e-9	1.75e-4	5.13e-2	3.76e-3	3.16e-9	1.80e-9	3.03e-9	2.79e-9	2.79e-4
5	L2	9e-16	9.4e-3	2.69	0.29	0	0	3.2e-7	0	6.9e-2
	fit	1.25e-9	1.75e-4	5.13e-2	3.24e-3	5..41e-11	1.75e-9	2.50e-9	1.25e-9	2.79e-4
10	L2	9e-16	9.4e-3	2.69	0.29	0	0	5.9e-7	0	6.9e-2
	fit	1.57e-9	1.75e-4	5.13e-2	3.23e-3	2.89e-9	4.50e-10	3.23e-9	3.00e-9	2.79e-4

the federated repaired model's weights into a vector V_{FedRep} and compare it with vector V_{Arachne} for the same $\mathcal{M}$ and Cl. Although the majority of results are identical, some L2 norms are non-zero, even when the set of suspicious weights are identical. However, the numerical errors are negligible in the overwhelming majority of cases, as even non-zero L2 norms are always close to 0.

Lines marked "fit" concern RQ2.2 and show the absolute value of the relative difference between fitness values of the corresponding repairs. We notice that they have orders of magnitude between 10^{-10} and 10^{-2}. These results show that the models repaired by FedRep perform comparably to those repaired by Arachne.

Answer to RQ2.1: FedRep produces slightly different repaired models to Arachne in terms of weight values, even when both methods target the same suspicious weights.
Answer to RQ2.2: Despite the imprecision introduced in FedRep, it still produces repairs of similar quality to Arachne.

RQ3: Influence of different data distributions. Finally, we report the results obtained by using FedRep under different data distributions. Table 3 shows the time (secs) to compute the fitness function for each client under various distributions. Each data distribution reports for each client c_i, what percentage of the data D^R is used by c_i. For example, "60-10-10-10-10" means that client 0 uses 60% of the repair data, and clients 1-4 each use 10% of it. For

Table 3: RQ3 – Aggregated fitness computation times (in seconds) for each client using different data distributions

(a) Distribution 80-5-5-5-5

	client 0	client 1	client 2	client 3	client 4
Max	8.26	0.52	0.38	0.46	0.55
Min	4.01	0.29	0.32	0.32	0.02
Avg	4.13	0.32	0.35	0.35	0.35

(b) Distribution 60-10-10-10-10

	client 0	client 1	client 2	client 3	client 4
Max	5.93	0.88	0.64	0.63	0.78
Min	3.11	0.54	0.56	0.55	0.57
Avg	3.58	0.58	0.59	0.59	0.62

(c) Distribution 40-15-15-15-15

	client 0	client 1	client 2	client 3	client 4
Max	4.92	0.90	0.89	0.90	1.06
Min	2.11	0.53	0.80	0.81	0.80
Avg	2.47	0.84	0.85	0.85	0.88

(d) Distribution 30-17.5-17.5-17.5-17.5

	client 0	client 1	client 2	client 3	client 4
Max	4.77	0.99	1.00	1.01	1.20
Min	1.55	0.00	0.93	0.93	0.95
Avg	1.62	0.94	0.97	0.97	0.99

each data distribution and each client, we report the maximum, minimum, and average time to compute the local fitness function. As expected, the client with the largest number of inputs is significantly slower at computing the fitness than other clients. In the worst case, when data is distributed according to a "80-5-5-5-5" split, such a client takes roughly four times longer than other clients to evaluate the fitness over its dataset (for all the DE individuals of one generation). This influences the overall runtime, as the search must happen synchronously, and can only proceed to the next generation when all clients' fitness scores have been received by the server. Hence, all clients must wait for the slowest one.

Answer to RQ3: Data imbalance between clients can impact FedRep's runtime, as the optimisation phase is synchronous, and a client with a larger dataset requires more time to compute the fitness of candidate solutions. Heterogeneity in clients' computing resources could however mitigate this.

6 THREATS TO VALIDITY

The validity of FedRep could be affected by different threats.

Construct validity. A threat of this type is that the metrics used to assess FedRep could be not suitable. Although we can prove the correctness of FedRep, numerical errors can introduce errors that lead to different results w.r.t. the non-federated repair of Arachne. Therefore, checking the difference between the results of FedRep and Arachne is a correct way of assessing the approach.

Conclusion validity. FedRep and Arachne both rely on the search algorithm DE that is affected by randomness. Thus, to check that FedRep and Arachne are indeed equivalent, for each experiment, we used the same seed. In the experiments, all the clients in FedRep are run on the same machine. This simplifies the experiment implementation, but does not influence the answer to the research questions. We leave a distributed implementation to empirically estimate other aspects (e.g, communication latency, the impact of heterogeneous architectures) as future work.

Internal validity. An internal threat could be to have a faulty implementation that leads to wrong conclusions. To mitigate this threat, the implementation of FedRep has been carefully tested.

External validity. It could be that the conclusions we draw in the experiments do not generalise. To mitigate this threat, we assessed it over VGG16, VGG19, and EnetB0. Although we experimented with only one dataset (BDD100K), we tried different distributions of the data among the clients (see RQ3).

7 RELATED WORK

This section introduces work related to the approach we present in this paper. First, it introduces efforts around DNN repair, before surveying works around federated computation and its applications.

DNN repair. As DNNs take a more central place in many safety-critical systems (e.g., autonomous driving) the impact of their faults grows. As such, the testing and debugging of DNN is an active topic [20, 32]. Once faults are identified through testing, the model must be improved to fix these faults. One way of correcting DNNs is to retrain (or finetune) them using inputs that have been identified as challenging to the model [7, 31]. In this work, however we focus on so-called *repair* techniques, that take inspiration from classic program repair, and combine *fault localisation* to identify faulty DNN components (e.g., neurons or weights), and optimisation techniques to fix those components.

Different repair techniques have been proposed [9, 18, 24, 26, 28], using different fault localisation and optimisation techniques. As Sect. 2 explains, we use Arachne [23] as the basis for our federated approach. Our approach could easily be adapted to various extensions of Arachne, such as that of Tokui et al. [26], that incorporates training data in the repair process to further prevent regressions, or that of Li Calsi et al. [13] that repairs multiple faults independently before merging the repairs to fix all faults. Further consideration would be required to adapt the federated approach to techniques targeting other types of repair such as Usman et al.'s [28] that targets aspects such as robustness and security by fixing backdoor security problems; or methods targeting other elements of the DNN such as the DNN's architecture rather than its weight values [18, 29] (see the survey by Kim et al. [11]).

Federated approaches. Federated learning [30] is a thriving research direction at the intersection of cryptography and deep learning that empowers several real-world applications. Hard et al. [8] use federated learning to train models for mobile keyboard predictions. The same principle favours the development of digital healthcare and telemedicine [21] without jeopardizing the patients'

privacy. Several works focus on the theoretical foundations of federated learning [3, 10], with some open questions still remaining. For instance, defending against data poisoning attacks [27] where attackers degrade the learning process by sending maliciously forged intermediate scores. Cao et al. [5] also studied poisoning attacks by several colluding users, and propose a scheme to mitigate the threat. Furthermore, federated learning inspired other learning techniques, such as Federated PSO [17], whose aim is a reduction of communication complexity by combining federation with PSO.

8 CONCLUSION

This work introduces FedRep, a DNN repair approach that allows different parties to cooperatively improve a DNN without the need for sharing raw data. It defines the distribution of both the fault localisation and weight optimisation processes used in the classic repair method Arachne and shows the theoretical equivalence of both methods. Experiments on three image classification DNN architectures using a large dataset in the domain of autonomous driving show that some differences are introduced in practice by imprecise floating point computations, but that FedRep produces a repair that is close to the Arachne's one and of the same quality.

ACKNOWLEDGMENTS

The authors are supported by Engineerable AI Techniques for Practical Applications of High-Quality Machine Learning-based Systems Project (Grant Number JPMJMI20B8), JST-Mirai. D. Li Calsi acknowledges funding from the Federal Ministry of Education and Research of Germany via the QDCamNetz project (Grant Number 16KISQ077).

REFERENCES

[1] 2023. Repository for the paper "Arachne: Search-Based Repair of Deep Neural Networks". https://github.com/coinse/arachne/tree/v2.
[2] bashtage. 2023. Implementation of ChaCha cipher-based PRNG. https://bashtage.github.io/randomgen/bit_generators/chacha.html.
[3] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1175–1191.
[4] Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Sina Noack, Hendrik Patzlaff, Hazar Harmouch, and Felix Naumann. 2022. The Effects of Data Quality on ML-Model Performance. *CoRR* abs/2207.14529 (2022). arXiv:2207.14529
[5] Di Cao, Shan Chang, Zhijian Lin, Guohua Liu, and Donghong Sun. 2019. Understanding distributed poisoning attack in federated learning. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 233–239.
[6] W. Diffie and M. Hellman. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* 22, 6 (1976), 644–654.
[7] Hazem Fahmy, Fabrizio Pastore, Mojtaba Bagherzadeh, and Lionel Briand. 2021. Supporting Deep Neural Network Safety Analysis and Retraining Through Heatmap-Based Unsupervised Learning. *IEEE Transactions on Reliability* (2021), 1–17.
[8] Andrew Hard, Chloé M Kiddon, Daniel Ramage, Francoise Beaufays, Hubert Eichner, Kanishka Rao, Rajiv Mathews, and Sean Augenstein. 2018. Federated Learning for Mobile Keyboard Prediction.
[9] Patrick Henriksen, Francesco Leofante, and Alessio Lomuscio. 2022. Repairing Misclassifications in Neural Networks Using Limited Data. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing* (Virtual Event) *(SAC '22)*. Association for Computing Machinery, New York, NY, USA, 1031–1038.
[10] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. 2021. Evaluating gradient inversion attacks and defenses in federated learning. *Advances in Neural Information Processing Systems* 34 (2021), 7232–7241.
[11] Jinhan Kim, Nargiz Humbatova, Gunel Jahangirova, Paolo Tonella, and Shin Yoo. 2023. Repairing DNN Architecture: Are We There Yet?. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 234–245.
[12] Davide Li Calsi, Matias Duran, Thomas Laurent, Xiao-Yi Zhang, Paolo Arcaini, and Fuyuki Ishikawa. 2023. Adaptive Search-Based Repair of Deep Neural Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Lisbon, Portugal) *(GECCO '23)*. Association for Computing Machinery, New York, NY, USA, 1527–1536.
[13] Davide Li Calsi, Matias Duran, Xiao-Yi Zhang, Paolo Arcaini, and Fuyuki Ishikawa. 2023. Distributed Repair of Deep Neural Networks. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 83–94.
[14] Davide Li Calsi, Thomas Laurent, Paolo Arcaini, and Fuyuki Ishikawa. 2024. Repository for the paper "Federated Repair of Deep Neural Networks". https://github.com/jst-qaml/federatedRepair.
[15] Andrei Mancu, Thomas Laurent, Franz Rieger, Paolo Arcaini, Fuyuki Ishikawa, and Daniel Rückert. 2024. More is Not Always Better: Exploring Early Repair of DNNs. In *2024 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*.
[16] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2022. Software engineering for AI-based systems: a survey. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–59.
[17] Sunghwan Park, Yeryoung Suh, and Jaewoo Lee. 2021. FedPSO: Federated Learning Using Particle Swarm Optimization to Reduce Communication Costs. *Sensors* 21, 2 (2021).
[18] Xuhong Ren, Jianlang Chen, Felix Juefei-Xu, Wanli Xue, Qing Guo, Lei Ma, Jianjun Zhao, and Shengyong Chen. 2022. DARTSRepair: Core-failure-set guided DARTS for network robustness to common corruptions. *Pattern Recognition* 131 (2022), 108864.
[19] Vincenzo Riccio, Gunel Jahangirova, Andrea Stocco, Nargiz Humbatova, Michael Weiss, and Paolo Tonella. 2020. Testing Machine Learning Based Systems: A Systematic Mapping. *Empirical Softw. Engg.* 25, 6 (nov 2020), 5193–5254.
[20] Vincenzo Riccio, Gunel Jahangirova, Andrea Stocco, Nargiz Humbatova, Michael Weiss, and Paolo Tonella. 2020. Testing machine learning based systems: a systematic mapping. *Empir. Softw. Eng.* 25, 6 (2020), 5193–5254.
[21] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. 2020. The future of digital health with federated learning. *NPJ digital medicine* 3, 1 (2020), 119.
[22] Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2019. Search Based Repair of Deep Neural Networks. *CoRR* arXiv:1912.12463v1 (2019). arXiv:1912.12463v1
[23] Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2023. Arachne: Search-Based Repair of Deep Neural Networks. *ACM Trans. Softw. Eng. Methodol.* 32, 4, Article 85 (may 2023), 26 pages.
[24] Bing Sun, Jun Sun, Long H. Pham, and Jie Shi. 2022. Causality-Based Neural Network Repair. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 338–349.
[25] Shogo Tokui, Susumu Tokumoto, Akihito Yoshii, Fuyuki Ishikawa, Takao Nakagawa, Kazuki Munakata, and Shinji Kikuchi. 2022. NeuRecover: Regression-Controlled Repair of Deep Neural Networks with Training History. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1111–1121.
[26] Shogo Tokui, Susumu Tokumoto, Akihito Yoshii, Fuyuki Ishikawa, Takao Nakagawa, Kazuki Munakata, and Shinji Kikuchi. 2022. NeuRecover: Regression-Controlled Repair of Deep Neural Networks with Training History. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1111–1121.
[27] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data poisoning attacks against federated learning systems. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*. Springer, 480–501.
[28] Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, and Corina S. Păsăreanu. 2021. NNrepair: Constraint-Based Repair of Neural Network Classifiers. In *Computer Aided Verification*. Springer International Publishing, Cham, 3–25.
[29] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. DeepLocalize: Fault Localization for Deep Neural Networks. In *Proceedings of the 43rd International Conference on Software Engineering* (Madrid, Spain) *(ICSE '21)*. 251–262.
[30] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. 2021. A survey on federated learning. *Knowledge-Based Systems* 216 (2021), 106775.
[31] Hao Zhang and W. K. Chan. 2019. Apricot: A Weight-Adaptation Approach to Fixing Deep Learning Models. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (San Diego, California) *(ASE '19)*. IEEE Press, 376–387.
[32] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2022. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering* 48, 1 (2022), 1–36.