Opportunistic Federated Learning: An Exploration of Egocentric Collaboration for Pervasive Computing Applications

Sangsu Lee†, Xi Zheng‡, Jie Hua†, Haris Vikalo†, Christine Julien†
†Department of Electrical and Computer Engineering, University of Texas at Austin
{sethlee, mich94hj, c.julien}@utexas.edu, hvikalo@ece.utexas.edu
‡Department of Computing, Macquarie University, james.zheng@mq.edu.au

***Abstract*—Pervasive computing applications commonly involve user's personal smartphones collecting data to influence application behavior. Applications are often backed by models that learn from the user's experiences to provide personalized and responsive behavior. While models are often pre-trained on massive datasets, *federated learning* has gained attention for its ability to train globally shared models on users' private data without requiring the users to share their data directly. However, federated learning requires devices to collaborate via a central server, under the assumption that all users desire to learn the same model. We define a new approach, *opportunistic federated learning*, in which individual devices belonging to different users seek to learn robust models that are *personalized* to their user's own experiences. However, instead of learning in isolation, these models opportunistically incorporate the learned experiences of other devices they encounter opportunistically. In this paper, we explore the feasibility and limits of such an approach, culminating in a framework that supports encounter-based pairwise collaborative learning. The use of our opportunistic encounter-based learning amplifies the performance of personalized learning while resisting overfitting to encountered data.**

***Index Terms*—pervasive computing, federated learning, collaborative deep learning, distributed machine learning**

I. Introduction

Smartphones, wearable devices, and other devices that fill pervasive computing environments are imbued with increasingly complex sensing, computation, and communication. However, applications still primarily rely on centrally located servers to support building and executing predictive models for real-time interactions. In this paper, we define *opportunistic federated learning*, which explores the potential for device-to-device collaboration to build expressive, accurate, and personalized models for use in pervasive computing applications.

The Setting. We consider pervasive computing applications that rely on models for classification, recommendation, or prediction. Examples include predicting the next word a smartphone user might type, classifying objects in a captured image, or predicting whether a captured photo is likely to be shared on social media. The training data for these models is *crowdsourced* – it is generated by a distributed set of independent devices. The derived models are potentially *personal*, both with respect to the fact that their outputs may be tailored to an individual and to the fact that the training data may be privileged, private, or proprietary. The goal is not for devices to converge to a single global model but rather for them to selectively collaborate to construct personalized goal models. The devices comprising pervasive computing environments are generally commodity smartphones, with on-board computing, storage, and wireless communication (e.g., WiFi and Bluetooth). They are resource constrained in energy, computation, and communication, yet they are capable of communicating directly with one another through opportunistic encounters.

Contributions. We introduce *opportunistic federated learning* and a novel approach to model sharing that we term *opportunistic momentum*. Devices belonging to individuals are bootstrapped with an initial model, which they *personalize* based on their experiences, as represented by the data collected by the device. When a device (the *learner*) encounters another device opportunistically (the *neighbor*), it uses a summary of the neighbor's available data to determine whether asking for learning support is (1) beneficial, based on the similarity (or dissimilarity) in their training data and learning goals and (2) feasible, based on the expected duration of the potentially fleeting encounter. The latter is predicted based on real-time information about mobility patterns and other context. If model collaboration is likely beneficial and feasible, the devices opportunistically exchange model gradients to generate a new local model for the learner. We present our approach as an egocentric one; of course it is possible that both devices in a pair can benefit from the assistance of the other, and an encounter may be used to support both participants as learners, depending on communication and computation constraints. Concretely, the paper's novel contributions are:

- We introduce *opportunistic federated learning* as an architectural pattern for learning from encounters and *opportunistic momentum* as an algorithmic tool for incorporating experiences of others.
- We examine the feasibility of opportunistic federated learning with respect to realistic differences in data distributions in pervasive computing networks.
- We examine the practicality of opportunistic federated learning with respect to the duration of encounters in pervasive computing applications.

In the long term, opportunistic federated learning will be one piece of a larger ecosystem in which cloud, edge, and

978-0-7381-4350-7/21/$31.00 ©2021 IEEE

opportunistic interactions are used in concert based on the instantaneous network conditions and application requirements.

II. Motivation and Related Work

Motivating Applications. Pervasive computing is teeming with applications that benefit from machine learning but for which training data is inherently private. These applications are often best served by *personalized* models. We focus on problems where the training task is *self-labeling*, including applications like keyboards that predict the next emoji the user will select based on the text they have typed [29], activity recognition on smartphones [31], or predictions of the popularity of content in social networks [37]. Self-labeling data makes it possible for devices to generate training data on-the-fly as part of a user's normal interaction with their device.

Different devices may have different experiences and generate widely varying data. We capture this diversity as a skew in the devices' *data label distributions*, i.e., the fraction of each label that a device "sees" [25, 33]. A car driven primarily on local roads may have very few images of semi-trucks in its data set, while a long-haul truck may have few samples of bicycles or children playing. An pedestrian application that recognizes landmarks might collect images of trees, mailboxes, and stop signs in the suburbs; the same application in a city center might see traffic lights, large buildings, and road signs. In emoji prediction, the emojis used by a teenager are likely to be very different from those used by a middle-aged adult.

Different devices may also have different *goal distribution*, i.e., the subset of labels the device wants to be able to classify correctly. While some goal distributions may be the entire set of labels, many goal distributions will be a subset of the label space. An object recognition system for vehicles may need to learn the entire label set for safety reasons. A landmark recognition system's goal distribution may be identical to its data distribution. And a user's emoji goal distribution may include any emoji used by others of a similar demographic.

Mobile devices are capable of on-device training but they do not want to share raw data. However, they may learn from neighboring devices that they encounter opportunistically. The characteristics that underlie our target applications are:

- *data distribution diversity*: the data one device encounters often differs from the data other devices encounter
- *goal set diversity*: two devices' goal distributions may be very different and may differ from the data distributions
- *encounter benefit*: devices benefit from opportunistic collaboration, but the benefits depend on overlaps between the devices' goal and data label distributions
- *data privacy*: devices are not willing to share their raw data with other devices they encounter opportunistically

Background and Related Work. As the capabilities of devices and the desire for data privacy have increased, federated and decentralized learning have emerged. We take for granted that deep learning has already moved into the pervasive computing world and focus here on efforts that go beyond applying inference to also enable some form of learning within the pervasive computing devices.

In federated learning, devices collaborate to construct a global model in a way that enables each individual device to maintain the privacy of its own data [21, 28]. Classically, a central coordinator orchestrates the process by delivering a model to each remote device, collecting and aggregating devices' contributions to training that model, then generating and distributing an updated model to continue the process. There are many applications of federated learning in pervasive computing. Wake word detection, also known as keyword spotting in smart home voice assistants, can use federated learning, protecting the potentially private audio data collected at users' end devices [24]. One of the most classic federated learning applications is next word prediction on a mobile device keyboard [11]. Still other applications have explored on-device image classification and image processing [35].

Recent work has also explored distributing the federated learning task across a hierarchical edge network [13] or selecting a coordinator from among a set of fog nodes [36]. These approaches are driven by a single coordinator and aim to learn a single global model. This differs from our goal, in which the effort is distributed and opportunistic, with each device operating egocentrically to improve its own model.

Related efforts in decentralized and personalized learning remove the coordinator. Some approaches frame the goal as a distributed consensus problem and rely solely on peer-to-peer interactions to disseminate model updates [6, 15]. Others personalize federated learning on mobile devices [16], even when users are expected to have diverse learning goals [34]. These efforts personalize local models that optimize a client's model against its own dataset, while still contributing to the training of a shared global model. Others have used collaboration among devices to improve local learning [32]; these approaches place significant constraints on collaboration or make strong assumptions about predictable contacts.

In simultaneously performing federated learning for a global goal and personalization for a local one, existing work has each client solve an optimization problem over its local data using a hyper-parameter that specifies the trade-off between the accuracy of the global and local models [7]. Alternatively, meta-learning can be used to adapt a global model to a local dataset [9]. These efforts are still based on a traditional federated learning backbone with a central server. Finally, recent work clusters clients with similar data distributions [27] or similar local updates to the global model [3] and trains a group model. Though these approaches provide some improvement with respect to both the global and local models [10], they are less applicable to pervasive computing, where clients are moving and their communications is opportunistic.

Even in federated learning, sharing model parameters or gradients potentially reveals something about private data, and efforts exist to attempt to reverse engineer these abstractions and recover sensitive information [39]. In practice, however, these techniques are limited, and the working consensus is that federated learning provides increased privacy relative to centralizing raw data [14]. In addition, privacy is not the only reason to employ federated learning; sharing model gradients

can incur reduced communication costs relative to directly sharing the raw data [21].

III. Opportunistic Federated Learning

We focus on applications that rely on deep learning models for prediction or classification tasks in which each device personalizes the model for its own use [8, 16, 34]. In contrast to prior work, we examine the benefits of incorporating opportunistic collaboration. Similarly to federated learning [28], collaborating peers desire to protect their raw data and instead share only snapshots of learned models.

Fig. 1 shows an overview of our framework. We start with a pre-trained generic model that we tailor using task-specific data to create a *bootstrap model* that is distributed to participating devices. Devices are controlled and carried by individual users, and they generate or collect (labeled) data. This local data is used to continuously fine-tune (i.e., personalize) the local model. When a device (the "learner") encounters another device ("the neighbor"), it may request the neighbor to perform a round of training on the learner's model using the neighbor's data. The learner sends its model parameters to the neighbor; the neighbor trains the model with its local data and returns the gradients, which the learner incorporates into its personal model. Though we assume that neighboring devices will participate, there is plenty of work on incentivizing collaboration in opportunistic environments [18, 19], which we can adopt in our scenarios in future work.

We describe the procedure for deciding whether to initiate this process in more detail below; as a preview, it is based on two inputs: (1) a comparison of the neighbor's data label distribution with the learner's goal distribution and (2) a prediction of the expected duration of the encounter. For the second, we rely on a long history of prior work in mobility and contact prediction [38, 4].

A. Data Label Distributions and Goal Distributions

We assume a set of N devices $\boldsymbol{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_i, \ldots, \mathcal{C}_N\}$, each with its own local data set, $\mathcal{D}_i$. In our example applications, these data sets are generated when the device takes images of its surroundings, shares images to social media, or sends text messages with emojis. Each sample is associated with a label (e.g., a photo may be labeled with a landmark or object within it; another photo may be labeled with whether it was shared on social media; a sequence of words may be labeled with the emoji that follows them). We capture device $\mathcal{C}_i$'s *data label distribution* ($\mathcal{L}_i$) as the relative frequency of each label within the local data set. The goal of opportunistic federated learning is to learn a local model for some task, i.e., to learn a model that can correctly label a novel input. Each device has a *goal distribution* ($\mathcal{G}_i$) of labels that it desires for its model to be successful at classifying. It is common that a given device's goal and data label distribution differ, and any two given devices may have different goal distributions.

B. Opportunistic Federated Learning

Each device $\mathcal{C}_i$ constructs its own local model, w_i. Because the local model changes over time based on the device's local training and encounters, we indicate a local clock t_i associated with $\mathcal{C}_i$'s model ($w_i^{t_i}$); we increment t_i every time $\mathcal{C}_i$'s local model is updated. $\mathcal{C}_i$'s bootstrap model is w_i^0.

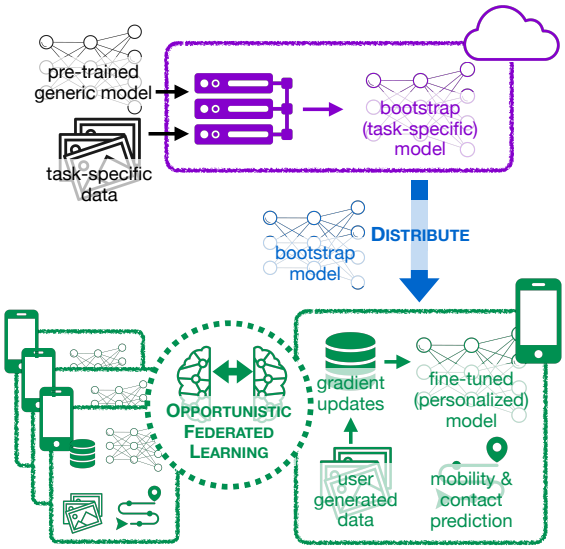


Fig. 1: System Architecture. The process optionally starts with a pre-trained model, which is specialized using task-specific data into the *bootstrap model* that is distributed to participating devices. Devices continuously fine-tunes a personalized model using locally collected data. Each device also uses on-device mobility and contact prediction to help decide when to engage in *opportunistic federated learning* exchanges with encountered devices.

Opportunistic federated learning updates the local model based on a combination of the device's local data and gradients obtained from encounters in order to optimize the local model for the goal distribution $\mathcal{G}_i$. Formally:

$$\min_{\boldsymbol{E_i}} \left\{ \sum_{t_i=0}^{|\boldsymbol{E_i}|} \ell(w_i^{t_i}; \mathcal{D}_{\mathcal{G}_i}) \right\}, \tag{1}$$

where $\boldsymbol{E_i}$ refers to the set of encounters that $\mathcal{C}_i$ has, and $\mathcal{D}_{\mathcal{G}_i}$ is a hypothetical data set with a data label distribution that satisfies the goal distribution $\mathcal{G}_i$. Intuitively, we strive to minimize the loss of a model that is learned from incorporating training across the encounters in $\boldsymbol{E_i}$.

The workflow is shown in Algorithm 1. All devices participate in continuous neighbor discovery [17, 20], through which they opportunistically discover nearby devices. The framework relies on three things: a similarity metric, a contact duration prediction, and a mapping of labels to learned gradients.

The algorithm first computes the similarity between the learner's goal distribution and the neighbor's data label distribution to determine whether the encounter will provide a useful learning opportunity. While there are many sophisticated metrics for similarity, our distributions are not wildly diverse, so we opt for a relatively simple metric. Specifically:

$$sim(\mathcal{P}_1, \mathcal{P}_2) = \sum_{l \in L} \min(\mathcal{P}_1(l), \mathcal{P}_2(l)). \tag{2}$$

Intuitively, if a label appears in both distributions, the sum includes the minimum frequency of that label in the two

Algorithm 1: Opportunistic Federated Learning

1 $sim(\mathcal{P}_1, \mathcal{P}_2)$: similarity metric for label distributions
2 $contact(t_i, \mathcal{C}_i, \mathcal{C}_j)$: predicted duration of encounter with device j at time t_i
3 $\Gamma[\mathcal{L}]$: maps a subset of labels to the most recent encounter gradient trained on those labels
4 **Function** ONDISCOVER: $\mathcal{L}_j$
5 **if** $sim(\mathcal{G}_i, \mathcal{L}_j) > \tau$ **then**
6 $w' = w_i^{t_i}$
7 **for** $r \leftarrow 0$ **to** ρ **do**
8 request computation of $\nabla\ell(w'; \mathcal{D}_j)$
9 $\Gamma[\mathcal{L}_j] \leftarrow \nabla\ell(w'; \mathcal{D}_j)$
10 $w' \leftarrow$ AGGREGATEGRADIENTS
11 **end**
12 $w_i^{t+1} \leftarrow w'$
13 **end**
14 **end**

distributions. If two distributions have no labels in common, the similarity score will be 0; if the two distributions are identical, the similarity score will be 1.

Algorithm 1 also relies on a predicted contact duration between two devices $\mathcal{C}_i$ and $\mathcal{C}_j$ at time t_i. This is provided by the underlying system, based on a system-level algorithm on each device that uses the device's context information to determine a likely length of contact. The specific implementation is outside the scope of this paper; we assume an off-the-shelf method to estimate contact duration [38, 4].

Thirdly, Algorithm 1 uses a data structure, Γ, that maps subsets of the label space (the keys) to gradients learned from encounters (the values). This structure is initially empty, but as the learner encounters and interacts with neighbors, it fills up with mappings from each label subset to the most recent gradients learned on that label subset. For instance, if the neighbor's data label distribution contains labels A, B, C, the learner will map the subset A, B, C to the final gradient returned from the learner in Algorithm 1.

When a device discovers a neighbor, the devices immediately exchange data label distributions. Each device independently executes the ONDISCOVER function that starts on line 4 of Algorithm 1. The local device (the learner) compares the neighbor's data label distribution to its own goal distribution using Equation 2. If the similarity is greater than a threshold (τ), the learner decides to engage with the neighbor to perform a session of remote training. This session comprises a customizable ρ number of rounds; ρ may be dictated by the task and underlying model, by the expected duration of the contact, or by some combination. If the pair of devices needs to split the duration of the encounter to perform the exchange in both directions, ρ might also be limited and negotiated.

Each round within a session (lines 8-10 in Algorithm 1) has two steps. First, the learner (i) requests remote learning from the neighbor (j). The learner sends the neighbor a copy of its current model by sending a summary of the model parameters. The neighbor loads the model and uses its own local data to compute the gradient $\nabla\ell(w'; \mathcal{D}_j)$, which it returns to the learner. The learner stores the returned gradient in the Γ data structure, mapped to the label set $\mathcal{L}_j$. If a mapping to $\mathcal{L}_j$ already exists, it is replaced. The learner applies one of a suite of gradient aggregation algorithms (described below), including a round of training on its own local data. These actions update the local model, which is used in any remaining rounds (i.e., the updated model is sent to the neighbor to repeat steps 8-10 in Algorithm 1). When the algorithm has completed ρ rounds, the learner's local model is updated, and the device is ready for the next encounter.

C. Aggregating Encounters

We implement two general options for filling in the AGGREGATEGRADIENTS function in line 10 of Algorithm 1 to update the local model. We refer to the first of these as *greedy aggregation*. Simply put, greedy aggregation directly averages in the gradients learned by the neighbor after incorporating one round of local learning. Formally, the update to the model w' in line 10 of Algorithm 1 is computed as:

$$\frac{\omega_{(\mathcal{L}_i,\mathcal{G}_i)}(\nabla\ell(w'; \mathcal{D}_i)) + \omega_{(\mathcal{L}_j,\mathcal{G}_i)}(\nabla\ell(w'; \mathcal{D}_j))}{\omega_{(\mathcal{L}_i,\mathcal{G}_i)} + \omega_{(\mathcal{L}_j,\mathcal{G}_i)}}, \tag{3}$$

The two gradients (the local one and the one from the neighbor) are both weighted with respect to the similarity of the corresponding data label distribution with the goal distribution. Building on existing work that similarly uses weights to address unbalanced data [7], the weights are:

$$\omega_{(\mathcal{L},\mathcal{G})} = \exp(-\lambda \times (1 - sim(\mathcal{G}, \mathcal{L}))), \tag{4}$$

where λ reflects how a model is prone to overfitting to a dataset that is small relative to the total number of labels. It can also be interpreted as a model's preference for a highly balanced dataset. For our experiments, we obtained λ empirically by running a series of personalization rounds using subsets of the training data used for the bootstrapped model.

This approach learns quickly from a device's encounters. On the other hand, when the data encountered is unbalanced with respect to the goal distribution, the model can overfit at the expense of the labels it encounters less frequently. Our second approach addresses this by computing a windowed average over a diverse set of recent encounters, where the diversity is determined by differences in the data label distributions of the contributed gradients. Every exchange with an encountered neighbor generates an update to Γ, which maps a neighbor's data label set to the gradients learned from that set. During a new encounter, this approach averages over *all* of these stored gradients before generating the update to w'. While this slows the speed of learning, it provides increased stability, especially when the learner encounters highly unbalanced data label distributions. We term this approach *opportunistic momentum*. Formally, the update to w' is computed as:

$$\frac{\omega_{(\mathcal{L}_i,\mathcal{G}_i)}(\nabla\ell(w'; \mathcal{D}_i)) + \left(\sum_{(\mathcal{L},\nabla)\in\Gamma} \omega_{(\mathcal{L},\mathcal{G}_i)}\nabla\right)}{\omega_{(\mathcal{L}_i,\mathcal{G}_i)} + \sum_{(\mathcal{L},\nabla)\in\Gamma} \omega_{(\mathcal{L},\mathcal{G}_i)}}. \tag{5}$$

The first term of the numerator accounts for the (weighted) contribution of a round of training on $\mathcal{C}_i$'s local data. The second term sums the gradients stored in the non-empty entries

in Γ, each weighted based on similarity. Before evaluating our approaches, we examine one final concept in our framework, the notion of decay with respect to the learning rate.

D. Learning Rate and Decay

Appropriately tuning the learning rate is important to avoid overfitting. We dynamically tune the learning rate by utilizing the concept of *decay*, which is common in deep learning [1]. Because our approach is completely decentralized, the learning rate for each device evolves independently. The learning rate at time t_i for device $\mathcal{C}_i$ is $\eta_i^{t_i} = \eta\alpha_i^{t_i}$, where η is an initial learning rate and $\alpha_i^{t_i}$ is the decay, computed as:

$$\alpha_{t_i} = \frac{\exp(\kappa \times (\phi - \left\|w_i^0 - w_i^{t_i}\right\|_2))}{\exp(\kappa \times (\phi - \left\|w_i^0 - w_i^{t_i}\right\|_2)) + 1}, \tag{6}$$

where

$$\alpha_{t_i} < min\{\alpha_0, ..., \alpha_{t_i-1}\},\ 0 < \phi,\ 0 < \kappa, \tag{7}$$

and ϕ and κ are constants. Opportunistic federated learning avoids overfitting to continuous encounters with a heavily skewed dataset by making an assumption that the minimum for a personalized task exists somewhere not too far away from the bootstrap model on the loss surface. The decay factor α is a sigmoid function, where an L2 distance from the initial weight is scaled and used as an input. This design encourages $\mathcal{C}_i$'s model to find a solution near the bootstrap model, as we assume it ensures a certain level of performance for all labels. As the model becomes more personalized, the learning rate decreases proportionally to the decay factor to seek a more fine-grained solution. At the same time, we only take the minimum of the decay factor to prohibit the model from reverting completely back to the original solution (the bootstrap model). The values of η, ϕ and κ are global constants determined prior to bootstrapping; like λ above, we determined the values by running experimental training with a subset of training data used for bootstrapping the initial model.

IV. Evaluation

We benchmark and evaluate our opportunistic federated learning framework in two threads. First, we present controlled experiments in which we manipulate devices' encounter patterns and data distributions in order to learn about and demonstrate how these impact performance. We the use realistic scenarios to demonstrate how opportunistic federated learning might perform in more realistic scenarios.

A. Datasets and Models

Opportunistic federated learning requires training models on commodity mobile devices. Further, training must be completed within the timeframe of an encounter between two neighboring devices. For these reasons, the models most suitable for opportunistic federated learning are likely to be relatively small and lightweight tasks. Our evaluation relies on two classification tasks; MNIST and CIFAR-10. In future work, we will explore pushing opportunistic federated learning even more, with additional models and with models that grow in size and complexity. In MNIST [23], the task is to correctly label images of handwritten digits 0 through 9. We replicate a "2NN" model from [28], which was used to prove centralized federated learning empirically. The network is composed of two fully-connected hidden layers, each with 200 neurons and ReLU activations. Our second dataset is CIFAR-10 [22], where the task is to recognize objects in images. We use a convolutional neural network (CNN) model, which is 11 layers deep with convolutional, max pooling, and dropout layers. MNIST and CIFAR-10 have 60,000 and 50,000 training images respectively. We used 10% and 25% of the entire training set, respectively, to train the bootstrap model and used the remainder of the data to create the devices' local datasets.

We chose these datasets because (1) they map to our motivating applications; (2) the models can realistically be trained on resource-constrained devices and (3) the datasets are sufficiently large. There are many applications that satisfy the first two constraints, but the third is more difficult to realize, in particular because our evaluation demands the ability to distribute the data in a skewed way among many devices.

B. The Feasibility of Encounter-Driven Learning

Opportunistic federated learning relies on coordinated rounds of device-to-device exchanges that occur when users' mobile devices encounter one another. It is essential to fit the execution of the exchange within the duration of an encounter. In particular, Lines 8-10 of Algorithm 1 unfold as (1) the learner sends the model; (2) the neighbor performs one round of training; (3) the neighbor returns the gradients; (4) the learner performs one round of training and aggregation. These four steps are repeated ρ times; the total needed time is:

$$t_{enc} = \rho \times (2t_{send} + 2t_{train} + t_{agg}). \tag{8}$$

To compute t_{train} and t_{agg}, we measured the computation time on a Raspberry Pi 4 (which has computational capabilities comparable to a smartphone). Training takes, on average, 1.543s and 5.74s for MNIST and CIFAR-10, respectively. For all approaches other than *opportunistic-momentum*, t_{agg} is 0. In *opportunistic-momentum*, Line 10 of Algorithm 1 requires iterating over the table Γ. The table has, at most, an entry for every subset of the goal set; however in practice the table is much smaller because it does not include entries that are completely subsumed by another and because a learner does not encounter all possible subsets of the goal set. To compute t_{agg}, we assumed the worst case $|\Gamma| = 2^{|\mathcal{G}|}$, or $|\Gamma| = 32$, given that, in all of our experiments, the size of the goal distribution set is 5. Measured empirically on the Raspberry Pi 4, the worst case t_{agg} for MNIST and CIFAR-10 are 0.064s and 0.448s, respectively, assuming Γ is loaded in memory.

Finally, computing t_{send} requires knowing the size of the model and the communication rate of the wireless channel; our MNIST model is 778KB (199,210 parameters), while CIFAR-10 is 4.8MB (1,250,858 parameters). Assuming two devices are connected via WiFi-direct, whose datarate is 250Mbps, t_{send} for MNIST is 0.020s and for CIFAR-10 is 0.153s. With

TABLE I: Required encounter durations ($|\Gamma| = 32$, $\rho = 6$).

	t_{train}	t_{agg}	t_{send}	t_{enc}
$\text{MNIST}_{\text{WIFI}}$	1.543s	0.064s	0.020s	19.14s
$\text{MNIST}_{\text{Bluetooth}}$	1.543s	0.064s	3.05s	55.50s
$\text{CIFAR-10}_{\text{WIFI}}$	5.740s	0.448s	0.153s	73.40s
$\text{CIFAR-10}_{\text{Bluetooth}}$	5.740s	0.448s	19.1s	300.77s

a lower datarate Bluetooth connection (i.e., 2Mbps), the t_{send} values are 3.05s and 19.1s, respectively.

Table I shows t_{enc} for both models. Many encounters in pervasive computing environments will satisfy these required durations, especially with a WiFi-direct connection (e.g., standing in line at the grocery store, chatting with a friend on the street, etc.). The longer durations needed when Bluetooth is used limit the usable encounters, but there are still many pervasive computing encounters that fall within this range (e.g., commuting on public transportation, eating in a restaurant, or sitting in a meeting). For our simulations we use a datarate of 1Mbps (a not-quite-ideal Bluetooth connection).

C. Evaluation Platform

We implemented our framework and algorithms in Python using TensorFlow [26] and Keras [5].[1] The models can run on resource constrained mobile devices. For the purposes of this paper, we also created a simulation environment that simulates each device's instance of the framework separately. The simulation environment provides an implementation of the "device-to-device" communication by passing messages between the threads. It also simulates the contact patterns that drive the encounters between the simulated devices.

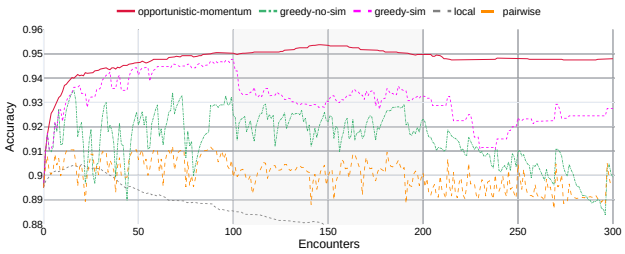
Determining whether to engage in an exchange has two components: (1) whether it is likely beneficial, based on the similarity between the data label and goal distributions and (2) whether it is feasible based on the predicted encounter duration. For the former, we use the similarity as computed in Equation 2. For the latter, we assume that predictions of contact duration from the underlying system are perfect; relaxing this assumption is left for future work. Given the predicted duration, we use Equation 8 for the amount of time required to complete an encounter.

D. Controlled Experiments

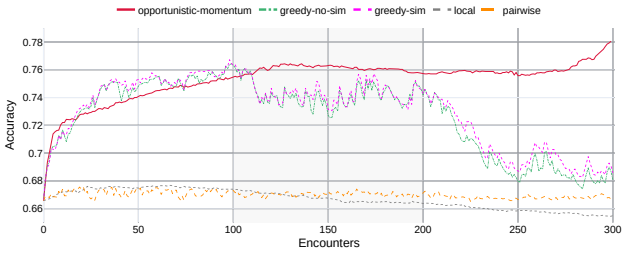
In our first experiments, we tightly controlled encounters and data distributions so that we could carefully benchmark our framework. We performed extensive evaluations on both datasets; due to space constraints, we show just one example. We compare the performance of five approaches:

- *local*: the model trains only on the learner's local data; for comparison purposes, the model continues to train over time even though no new data is generated.
- *pairwise-fed-avg*: the model trains using a pairwise version of federated averaging [21]; a pair of devices perform as many rounds of pairwise federated averaging as each encounter duration allows, starting with a base model that is the average of the two devices' models, rather than from the learner's model as in the remaining approaches.

[1]https://github.com/UT-MPC/swarm



(a) MNIST



(b) CIFAR-10

Fig. 2: Comparing algorithm options for (a) MNIST and (b) CIFAR-10. The x-axis is the number of encounters the device has experienced. The y-axis is the model accuracy. Each shaded region is a shift in the data distributions on the encountered devices.

- *greedy-no-sim*: the model trains on every encounter without considering the similarity between the learner's goal distribution and the neighbor's data label distribution using the greedy aggregation from Equation 3 with $\omega = 0.5$ (i.e., equal weight to local data and neighbor's data).
- *greedy-sim*: we limit the training encounters to those of sufficient similarity ($\tau = 0.2$), still using $\omega = 0.5$.
- *opportunistic-momentum*: we train the model using the strategy in Equation 5 with $\tau = 0.2$, including weights determined according to Equation 4 and dynamic decay of the learning rate (Equation 6).

The first two models are baselines for comparison; the remaining three models are all novel contributions of our work. Our goal is to understand the conditions under which each is suitable for supporting opportunistic federated learning.

Fig. 2 shows results for both datasets. MNIST has a label for each digit in the range 0-9; in CIFAR-10 there is a label for each class of object recognized in a photo. For simplicity, we refer to both label sets with numbers 0-9. In Fig. 2, the goal distribution contains exactly five of the ten labels, specifically labels $\{0, 1, 2, 3, 4\}$. Every device has a local dataset of with 80 items in MNIST and 150 in CIFAR-10. The learner's local dataset contains equal numbers of labels $\{0, 1\}$. The encountered data label distributions change over time:

- *first 100 encounters*: 50% of the first 100 encounters are with neighbors that see exactly and only labels $\{2, 3\}$, and the other 50% are with neighbors that see three labels, selected randomly from all ten labels.
- *encounters 100-200*: in the middle, 50% of encounters are

with neighbors that see exactly and only labels $\{3, 4, 5\}$, and the other 50% are with neighbors that see only three labels, selected randomly from all ten labels.

- *encounters 200-300*: in the last period, 50% of encounters are with neighbors that see exactly labels $\{4, 5, 6\}$, and the other 50% are with neighbors that see only three labels, selected randomly from all ten labels.

A completely egocentric approach that trains only on a device's local data (*local* in Fig. 2a) results in a model that overfits very quickly. Pairwise federated averaging (*pairwise-fed-avg*), while intuitively promising, also suffers under these workloads (and in real environments). The reason is that federated averaging combines the models of the two devices to generate a new base model used for training. Because these devices have been working independently, their models have likely diverged. In contrast, the remaining approaches all use the learner's model as the base, even on the neighbor's device.

The next approaches are *greedy-no-sim* and *greedy-sim*. The former greedily incorporates whatever it can achieve with any encountered neighbor, without considering its goal distribution. In this particular example, *greedy-no-sim* performs the worst of our approaches because this example has a very unbalanced and unstable distribution of encountered data. When the encountered data is more well mixed, *greedy-no-sim* performs better and is, in some cases, the best performing approach. The downside of *greedy-no-sim* is most apparent in the third region, when the likelihood that the device encounters data label distributions that overlap with its goal distribution decreases. In contrast, *greedy-sim* is more resilient to changes in encountered data label distributions because it only requests learning from devices whose distributions are sufficiently similar to the goal distribution. This example uses a value of $\tau = 0.2$ in Algorithm 1; even this minimal overlap has a substantial benefit to performance. Larger values for τ result in even better performance for *greedy-sim*, albeit in exchange for somewhat slower convergence.

In this example, *opportunistic-momentum* ended with the highest accuracy. *Opportunistic-momentum* is the most resilient to dramatic changes in data distributions, and it is best at avoiding overfitting to a distribution it sees for a period of time because it continuously integrates meaningful gradients that it previously collected, even if it does not continue to encounter them. The greedy approaches dip in performance as we move from one region to another because the models have overfit to the data label distributions. In contrast, *opportunistic-momentum* is quite resilient to sudden changes in data label distribution. Arguably, this is a contrived situation, designed to benefit *opportunistic-momentum* relative to other strategies. If the data distributions that the user encounters are highly overlapping with the goal distribution, the greedy strategies outperform *opportunistic-momentum*. For this reason, we now step into more real-world experiments, where the data distributions are much less controlled and contrived.

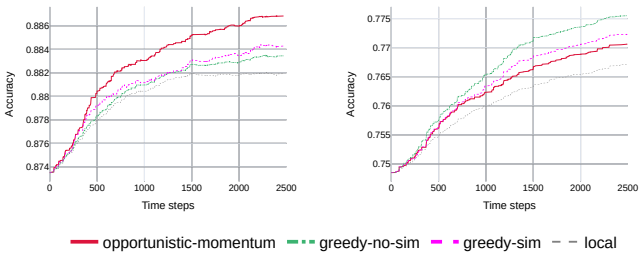


Fig. 3: Comparing algorithm performance for a real-world scenario for (a) MNIST and (b) CIFAR-10. The x-axis is the elapsed time of the simulation. The y-axis is the model accuracy.

E. Real World Scenarios

We next allow devices' movement patterns to evolve independently according to the Levy walk mobility model [2, 30], which, among stochastic mobility models, is known to capture human mobility well [2, 12]. Levy walk assumes that the majority of an individual's movements are in a small local area, with a few very large movements every once in a while. We opted for randomized mobility that allows more careful understanding of the impacts of mobility, which in turn enables a more careful benchmarking of the performance of our algorithm. Future work will include evaluation on collected mobility traces and on real devices.

We created a square space, divided into 9 equally sized regions. Each device is assigned an anchor region (e.g., the user's home), with five devices assigned to each region. We simulate *episodes*; in each episode, the user starts out at home, makes some trips away from home, and returns home at the end of the episode. Each runs consists of 10 episodes. Devices encounter one another by coming within a pre-defined communication range; some encounters are ephemeral, as the devices move past each other on their trajectories, while others are longer-lasting, as devices stay nearby for some period of time. We assume that devices have perfect knowledge of predicted encounter durations.

Each region is associated with two labels. Any device's whose anchor location is in that region has a data label distribution that contains exactly those two labels. Each device sets its goal distribution to be those two labels and three additional randomly chosen labels. As such, devices with anchor locations in the same region likely have different goal distributions. This distribution of data and goals mimics a real world environment where devices collect different data depending on their experiences but also have diverse goals, e.g., due to the fact that their different travel patterns lead them to encounter different data that needs to be classified.

Fig. 3 shows the results. For the simpler MNIST model, the framework performs as expected, given the results in the controlled experiments. For CIFAR-10, however, the best performing of our models is *greedy-no-sim*. In this scenario, the data label distributions and goal distributions have a significant random component to them. As a result, the labels are relatively well distributed among devices, putting our models in a situation similar to the far left region of Fig. 2. This benefit does come at a cost; because *greedy-no-sim* takes advantage

of every possible encounter, it has 18% higher overhead compared to *opportunistic-momentum*. This last observation opens a piece of future work: designing an approach that can adapt to the changing nature of surrounding data distributions. When the encountered data is evenly distributed, the algorithm can enter a *greedy-sim* mode, taking advantage of any and all opportunity for collaboration. When the algorithm senses an imbalance in encountered data, it could transition into *opportunistic-momentum*.

In conclusion, these results show that our approaches to opportunistic federated learning can consistently outperform local personalized learning and can be very resilient to overfitting and dramatic fluctuations in the encountered data distributions.

V. Conclusions and Future Work

This paper explored a novel paradigm for machine learning in pervasive computing, which we term *opportunistic federated learning*. We defined a framework through which devices can collaborate opportunistically with other devices in their surroundings using only device-to-device communication links. We defined an algorithm within the framework, *opportunistic momentum*, which provides a robust mechanism to continuously integrate learning from encounters in a way that improves over a well-informed greedy approach. Overall, our results demonstrate that there are real-world pervasive computing scenarios and applications that can garner significant benefits from this collaborative yet personalized approach to *in situ* training of reasonably coupled deep learning models.

Our results show the significant promise of opportunistic federated learning for diverse pervasive computing applications. Near term future work will extend our evaluation to even further understand the performance of the opportunistic-momentum algorithm, including assessing the impacts of increased varieties of goal distributions, evaluating even more models (e.g., human activity recognition models), and relying on real-world mobility traces. We should also consider what happens when a device's goal distribution changes, either gradually (e.g., as emoji trends come and go) or abruptly (e.g., because a user changes their job or home environment). It is possible that our approach will successfully adapt to gradual changes, but more abrupt changes will require re-bootstrapping a model, perhaps from an encountered neighbor.

References

[1] Y. Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[2] B. Birand et al. "Dynamic graph properties of mobile networks under levy walk mobility". In: *Proc. of MASS*. 2011.

[3] C. Briggs, Z. Fan, and P. Andras. *Federated learning with hierarchical clustering of local updates to improve training on non-IID data*. 2020. arXiv: 2004.11791 [cs.LG].

[4] H. Chen and W. Lou. "Contact expectation based routing for delay tolerant networks". In: *Ad Hoc Networks* 36 (2016), pp. 244–257.

[5] F. Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[6] I. Colin et al. "Gossip Dual Averaging for Decentralized Optimization of Pairwise Functions". In: *Proc. of ICML*. 2016.

[7] Y. Deng, M. Mahdi Kamani, and M. Mahdavi. *Adaptive Personalized Federated Learning*. 2020. arXiv: 2003.13461 [cs.LG].

[8] C. T. Dinh, N. H. Tran, and T. D. Nguyen. *Personalized Federated Learning with Moreau Envelopes*. 2020. arXiv: 2006.08848 [cs.LG].

[9] A. Fallah, A. Mokhtari, and A. Ozdaglar. *Personalized Federated Learning: A Meta-Learning Approach*. 2020. arXiv: 2002.07948.

[10] A. Ghosh et al. *An Efficient Framework for Clustered Federated Learning*. 2020. arXiv: 2006.04088 [stat.ML].

[11] A. Hard et al. *Federated Learning for Mobile Keyboard Prediction*. 2019. arXiv: 1811.03604 [cs.CL].

[12] S. Hong et al. "Routing performance analysis of human-driven delay tolerant networks using the truncated levy walk model". In: *Proc. of MobilityModels*. 2008, pp. 25–32.

[13] S. Hosseinalipour et al. *Multi-Stage Hybrid Federated Learning over Large-Scale D2D-Enabled Fog Networks*. 2020. arXiv: 2007.09511.

[14] Y. Huang et al. "Instahide: Instance-hiding schemes for private distributed learning". In: *Proc. of ICML*. 2020.

[15] F. Iutzeler et al. "Asynchronous distributed optimization using a randomized alternating direction method of multipliers". In: *Proc. of CDC*. 2013.

[16] Y. Jiang et al. *Improving Federated Learning Personalization via Model Agnostic Meta Learning*. 2019. arXiv: 1909.12488 [cs.LG].

[17] C. Julien et al. "BLEnd: Practical Continuous Neighbor Discovery for Bluetooth Low Energy". In: *Proc. of IPSN*. 2017.

[18] J. Kang et al. "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory". In: *IEEE Internet of Things Journal* 6.6 (2019).

[19] L. U. Khan et al. "Federated learning for edge networks: Resource optimization and incentive mechanism". In: *IEEE Communications Magazine* 58.10 (2020), pp. 88–93.

[20] Philipp H Kindt and Samarjit Chakraborty. "On optimal neighbor discovery". In: *Proc. of SIGCOMM*. 2019, pp. 441–457.

[21] J. Konečný et al. *Federated Learning: Strategies for Improving Communication Efficiency*. 2017. arXiv: 1610.05492 [cs.LG].

[22] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. University of Toronto, 2009.

[23] Y. LeCun et al. "Gradient-based learning applied to document recognition". In: *Proc. of the IEEE* 86.11 (1998), pp. 2278–2324.

[24] D. Leroy et al. "Federated learning for keyword spotting". In: *Proc. of ICASSP*. 2019, pp. 6341–6345.

[25] A. Li et al. *LotteryFL: Personalized and Communication-Efficient Federated Learning with Lottery Ticket Hypothesis on Non-IID Datasets*. 2020. arXiv: 2008.03371 [cs.LG].

[26] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: http://tensorflow.org/.

[27] Y. Mansour et al. *Three Approaches for Personalization with Applications to Federated Learning*. 2020. arXiv: 2002.10619 [cs.LG].

[28] B. McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Proc. of AISTATS*. 2017.

[29] S. Ramaswamy et al. *Federated Learning for Emoji Prediction in a Mobile Keyboard*. 2019. arXiv: 1906.04329 [cs.CL].

[30] I. Rhee et al. "On the Levy-Walk Nature of Human Mobility". In: *IEEE/ACM Transactions on Networking* 19.3 (2011), pp. 630–643.

[31] S. A. et al. Rokni. "Personalized human activity recognition using convolutional neural networks". In: *Proc. of AAAI*. Vol. 32. 1. 2018.

[32] P. Vanhaesebrouck, A. Bellet, and M. Tommasi. "Decentralized Collaborative Learning of Personalized Models over Networks". In: *Proc. of AISTATS*. 2017.

[33] D. C. Verma et al. "Approaches to address the data skew problem in federated learning". In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*. 2019.

[34] K. Wang et al. *Federated Evaluation of On-device Personalization*. 2019. arXiv: 1910.10252 [cs.LG].

[35] Y. Xiong, H. J. Kim, and V. Hedau. *ANTNets: Mobile Convolutional Neural Networks for Resource Efficient Image Classification*. 2019. arXiv: 1904.03775 [cs.CV].

[36] T. Zhang et al. "Achieving Democracy in Edge Intelligence: A Fog-based Collaborative Learning Scheme". In: *IEEE Internet of Things Journal* (2020).

[37] Y. Zhang and A. Jatowt. "Image tweet popularity prediction with convolutional neural network". In: *Proc. of ECIR*. Springer. 2019.

[38] H. Zhou et al. "Predicting Temporal Social Contact Patterns for Data Forwarding in Opportunistic Mobile Networks". In: *IEEE Transactions on Vehicular Technology* 66.11 (2017), pp. 10372–10383.

[39] L. Zhu, Z. Liu, and S. Han. "Deep leakage from gradients". In: *Proc. of NeurIPS*. 2019.

Decentralized Landslide Disaster Prediction for Imbalanced and Distributed Data

Ren Ozeki
Osaka University, Japan
Japan Meteorological Agency
r-ozeki@ist.osaka-u.ac.jp

Haruki Yonekura
Osaka University
Osaka, Japan
h-yonekura@ist.osaka-u.ac.jp

Hamada Rizk
Osaka University, Osaka, Japan
Tanta University, Tanta, Egypt
hamada_rizk@ist.osaka-u.ac.jp

Hirozumi Yamaguchi
Osaka University
Osaka, Japan
h-yamagu@ist.osaka-u.ac.jp

***Abstract*—Precise disaster prediction plays a critical role in saving lives. Traditional landslide prediction methods have predominantly relied on deep neural networks such as CNNs and LSTMs. However, these methods face two main challenges. The first is the class imbalance issue, as landslides are rare, disrupting the training process. The second challenge stems from decentralized data management, with variations in volume and characteristics across regions. Typically, local municipalities manage disaster data within their regions, and sharing or migrating this data is not straightforward. This paper presents *SlideSafe*: a novel landslide prediction system that combines spatio-temporal contrastive learning and collaborative learning[1]. It begins by training a contrastive learning model to extract meaningful representations of land characteristics in each region. Subsequently, these trained models are merged among regions with similar characteristics, leveraging federated learning. The federated models are then fine-tuned and customized for the landslide event prediction using the data specific to each region. Experimental results indicate that the proposed system achieves higher precision under 100% recall compared to state-of-the-art federated learning methods, which are often adversely affected by non-iid data and data scarcity.**

***Index Terms*—Decentralized Landslide Prediction, Class imbalance, Contrastive Learning, Selective Federated Learning**

I. Introduction

Accurate disaster prediction is essential for safeguarding lives through disseminating warnings, providing vital time for evacuation, and directing people to secure locations. Global climate change has heightened the need for improving global disaster management systems. Specifically, rainfall-induced landslides are particularly vulnerable to climate change, potentially occurring in historically safe areas. Unlike other disasters, predicting landslides is challenging due to complex factors like geological conditions, terrain, rainfall patterns, and soil composition, requiring urgent attention [1].

Traditionally, meteorological agencies worldwide have relied on statistical data processing from numerical weather models and observation systems for landslide prediction [2]. These traditional approaches frequently encounter challenges due to their incapacity to adequately account for the intricate spatial interdependencies and the evolving dynamics of water accumulation—both of which play crucial roles in landslide occurrences. In light of these constraints, existing methods have turned to harnessing the power of deep neural networks, including Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks [3], [4].

Predicting landslides practically presents a substantial challenge due to the scarcity of labeled data and class imbalance in datasets. Landslide events are rare, leading to an unequal distribution of positive and negative cases. For instance, in a dataset of weather-related incidents, there might be thousands of records of non-landslide events but only a handful of landslide occurrences. This class imbalance greatly hinders the training of machine/deep learning models, often resulting in a biased model that does not consider the minority class, which in this case, represents landslides. Due to the problems of class imbalance and the scarcity of labeled data, current methods do not attain nearly 100% recall while maintaining a reasonably low false positive rate, a critical requirement for enabling residents to respond effectively to disasters [5].

Another challenge arises due to the decentralized nature of data management and data disparities among different regions. Federated learning has emerged as a promising solution for model training over distributed data. This can be attributed to the fact that landslides and other disasters are typically monitored and managed by local municipalities, which means that the relevant disaster data is stored with these municipal authorities. To be more specific, local municipalities require region-specific data and have established their own monitoring facilities. However, the data collection processes employed by local municipalities may not be consistent, resulting in variations in data quality and quantity. Besides, terrain characteristics and weather conditions vary significantly from one region to another. These heterogeneities lead to a non-independent and non-identically distributed (non-IID) data problem, which has been demonstrated to impact the efficiency of the training process of federated learning in a recent study [6], [7].

To address these challenges, we propose *SlideSafe*: a novel decentralized landslide prediction system. *SlideSafe* involves the use of contrastive learning within each municipality to capture spatio-temporal dependencies and patterns from imbalanced and distributed data. Contrastive learning has garnered significant attention in recent years for its ability to effectively learn high-quality representations from complex input data, irrespective of label information. Subsequently, we quantify the similarity between each pair of municipalities based on

[1]Our implementation is available here (https://github.com/mclab-osaka/slidesafe)

979-8-3503-2603-1/24/$31.00 ©2024 IEEE

their encoded features. This similarity measurement guides a decentralized collaborative learning process where municipalities with similar features collaborate to construct a shared model (an encoder) to enhance the learning process. Finally, we fine-tune the shared model to adapt it to each municipality for accurate landslide prediction.

The proposed system underwent a thorough evaluation using real-world data from government agencies in Japan. The data was collected over two years to assess the system's effectiveness in precision under 100% recall. The obtained results validate the system's capability to achieve a remarkable landslide prediction performance of 62.4%. This represents a substantial improvement over the state-of-the-art techniques, surpassing them by up to 40.4%. These results demonstrate the feasibility of predictive accuracy in real-world applications.

II. Related Work

A. Landslide prediction

Traditional landslide prediction approaches are algorithm-based methods [2], [8]. For instance, the author of [2] proposed a statistical method based on rail fall amount to issue early warning of landslides and evaluate their method using a dataset of India as a case study. However, since these works don't consider the spatio-temporal dependency of landslides, like soil distribution, elevation, and rainfall-related water accumulation, these methods have significant limitations in accuracy and generality regarding regions.

To capture spatio-temporal dependency of landslides, machine learning methods are also proposed [3], [4]. To predict landslides accurately by capturing spatial dependency, the author of [3] tries to predict landslide events with CNN. The literature [4] tries to predict the displacement of the ground that is a sign of landslide by utilizing LSTM.

Although these existing approaches are promising in terms of accuracy, to operate a landslide prediction system officially, it is more important to keep the 100% recall to avoid a situation where residents can't evacuate. simultaneously disaster prediction systems need to keep high precision because people do not trust low-precision information. Since landslides are unusual and rare events, it is challenging to achieve high precision while keeping 100% recall in landslide prediction due to class imbalance. However, these existing researches don't consider this class imbalance problem and the specific requirements of disaster systems. *Unlike existing research, SlideSafe tackles the class imbalance problem in landslide prediction and the particular requirements of disaster systems.*

B. Machine Learning from Imbalanced Data

The class imbalance problem is one of the fundamental problems of machine learning [9]. For instance, in binary classification, the presence of majority and minority classes is assumed, with a specific imbalance ratio. Since the standard classifier is optimized by a loss function that treats both classes equally, the minority class is almost ignored. However, the minority class is more important to detect in usual cases that include our target research area. To overcome this class imbalance problem, many researches have addressed this problem [10]–[12]. Approaches to solving class imbalance problems are mainly categorized into two: Data-level approach and algorithm-level approach.

The data-level approach aims to obtain balanced training datasets from imbalanced dataset utilizing undersampling or oversampling [13]–[15]. Since undersampling methods risk removing data important for a model to learn the boundary, oversampling gains more attention than undersampling [14], [16]. To generate similar additional data even for multi-modal data, some data augmentation methods using generative models are proposed [16] because it is difficult for the traditional statical approach such as SMOTE [14] to generate additional high-quality multi-modal data. The algorithm-level approach works directly during the training procedure of the classifier, disliking the previous data-level approach. The most commonly addressed issue with the algorithm-level approach is loss function adaptation, such as Focal loss [17], cost sensitive learning [18] and Mean False Error [19]. Mean False Error [19] balances the weight of loss from minority and majority classes. Focal loss [17] reduces the impact of easy instances on the loss function.

While the data-level approach holds promise, the cost of data augmentation can be substantial, especially in scenarios with high-class imbalance ratios or multi-modal input data. Conversely, relying solely on loss function adaptation often proves ineffective when the classifier is a deep, complex neural network because the amount of minority data is insufficient for obtaining high-quality latent representation. *To obtain meaningful latent representation, SlideSafe employs contrastive learning as self-supervised representation learning that is robust to label imbalance [20].*

C. Federated learning

Federated learning (FL) is widely used when it is difficult to collect distributed data on a server due to privacy, security, or data migration costs because FL enables distributed clients to train a shared model collaboratively without exchanging local data [21]. However, it is known that traditional FL can't train efficiency when global data distribution and local data distribution differ, which is called non-IID [22]. To build an effective model for global distribution in non-IID settings, many existing works address FL in non-IID settings [7], [22]–[24]. FedProx [22] is designed to avoid the local model deviating greatly from the global model.

Although these works succeed in converging learning and improving the global model under non-IID data, the author of [25]–[27] proposed personalization in FL because one global model cannot fit all clients. One example is FedProx-FT [25], which refines the global model from FedProx [22] using local data to create personalized models. One of the similarity-based approaches is FedAMP [26], which keeps the personalized cloud model for each client trying to learn in each party without sharing each client's data. MOON [7] and FedMoCo [24] use federated learning only between similar nodes whose

representations are similar. This approach usually does not lead to model convergence and thus affects the system performance. Additionally, none of the above-mentioned techniques has been applied to the field of disaster. *In contrast, SlideSafe adopts a decentralized collaborative learning mechanism and neighbor-finding way based on weight similarity that ensures model convergence and thus saves in communication costs as demonstrated in [28].*

III. Problem statement

There are various types of land-related disasters, including landslides, mudslides, slope failures, surface/deep collapses, and debris flows resulting from heavy rainfall. In this paper, our primary focus is on *landslides triggered by rainfall*. Each local municipality, responsible for monitoring and administering landslide events, weather conditions, topographical features, and vegetation within its respective geographical area, shall henceforth be referred to as a "client". We define $\boldsymbol{X}^i_{a,b}$ and $y^i_{a,b}$ as the features and the label of client i during specific time period $[T_a, T_b]$, respectively. The notations $\boldsymbol{X}^i$ and y^i do not specify time periods, and $\boldsymbol{X}^i_t$ and y^i_t are the values in the specific moment t.

SlideSafe aims to predict landslide occurrence in the next time slot T_{t+1} where T_t is the current time. It takes into account two key input factors: time-series rainfall data and stable land characteristics that remain constant throughout the time period. The land characteristics encompass a range of features, including soil properties (categorized into 10 distinct classes), vegetation types (categorized into 11 classes), and elevation slope data(represented as real numbers with eight categorized directions). These land characteristics serve as an essential input for the system. Additionally, the system utilizes time series rainfall data, which consists of non-negative real numbers. This rainfall data is collected from government public agencies, ensuring the reliability and accuracy of the input information. As reported in a recent study [29], the latest numerical weather forecast is so accurate within a day that it can be considered actual value. Therefore, we can use the forecasted future rainfall amount data $X^i_{t+1,t+u}$ $(1 \leq u \leq \Lambda - t)$ as well as the past observation $X^i_{t-h,t}$ $(0 \leq h \leq t-1)$ as input for the prediction model. Our prediction task, aimed at client i for predicting the binary label y^i at time T_{t+1}, is formalized by Eq. (1). In Eq.(1), $\hat{y}^i_{t+1} \in \{0, 1\}$ represents the predicted binary outcome, indicating whether a landslide will occur at T_{t+1}. To achieve accurate landslide predictions for all clients' regions, *SlideSafe* minimizes the sum of all clients' loss ℓ, which is calculated using each client's model weight w_i, as expressed in Eq. (2).

$$\hat{y}^i_{t+1} = \mathcal{F}(\boldsymbol{X}^i_{t-h,t+u}) \tag{1}$$

$$\sum_{i \in [1,N]} \mathbb{E}_{z \sim D_i}[\ell(\boldsymbol{w}_i; \boldsymbol{X}^i)] \tag{2}$$

In this equation, $z = (\boldsymbol{X}^i, y^i)$ represents data following a probabilistic joint distribution D_i. Due to significant variations in landslide occurrences and related features across regions, the data distribution D_i is heterogeneous and non-IID (non-independent and identically distributed).

This approach is driven by several key advantages of decentralized collaborative learning, including ensuring regional information security and facilitating comprehensive knowledge sharing across different regions. This knowledge-sharing aspect enables the system to adapt to any regional changes influenced by climate change or other factors. By adopting decentralized collaborative learning, *SlideSafe* leverages these advantages to create accurate, robust, and adaptable prediction models for diverse geographical areas. Furthermore, it is worth noting that D_i exhibits class imbalanced data distribution, which poses challenges for traditional machine learning and federated learning approaches in obtaining general knowledge or personalized models, especially within a decentralized collaborative learning framework. For instance, clients with no prior experience in landslides may participate in federated learning, potentially negatively impacting the learning process. *Despite the distinctiveness of each client's distribution D_i, SlideSafe employs decentralized personalized collaborative learning. In the proposed approach, the prediction model leverages knowledge from other clients to acquire comprehensive insights about landslides, enabling generalization to potential future changes in the region while also ensuring specialization for the target client and its associated distribution.*

IV. *SlideSafe*– Landslide Predictor for Distributed Data

The objective of *SlideSafe* is to predict landslides using the model of each client in their respective regions. To preprocess raw data (such as the rainfall, soil information, and landslide occurrence points) into a manageable and interpretable format for machine learning models, the **virtual gridding module** converts all data into square grid cell formats. The formatted data is then input into the **feature extractor module** of each client to extract meaningful latent features from complex spatio-temporal input within their respective regions. This extraction is achieved using contrastive learning techniques, as described in [20], where training is conducted self-supervised. Contrastive learning is designed to reduce the distance between the representations of different augmented data of the same input (i.e., positive pairs), and increase the distance between the representations of augmented views of different input (i.e., negative pairs). Therefore, contrastive learning contributes to distinguishable latent representation even in a self-supervised manner. Then, clients communicate with each other to efficiently share their knowledge, which includes the weights of the contrastive learning models, through a **decentralized collaborative learning mechanism**. Within this mechanism, clients can identify and collaborate with other clients that have similar data distributions, enabling efficient collaborative learning. This mechanism identifies similar clients by measuring the similarity of the local model's update because the similar direction of model update helps convergence of

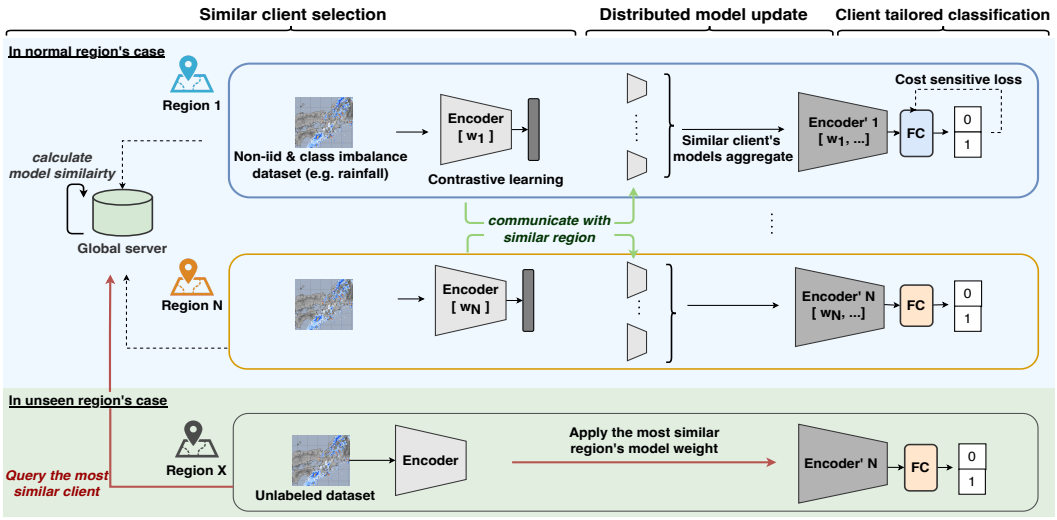


Fig. 1. *SlideSafe* overview in heterogenous environments

aggregated model and reflects similar data distribution as written in [28]. Finally, each client fine-tunes a shallow neural network, the **client-tailored classifier**, to predict landslides within their respective regions. This fine-tuning process utilizes the feature extractor trained in the previous module. We note that the virtual gridding module, feature extractor module, and client-tailored classifier are client-dependent functions that run on each client, while the decentralized, federated learning mechanism is run on a single global server. The architecture will be explained later in this section.

Since our focus is on a non-IID and class-imbalanced disaster dataset, *SlideSafe* needs to obtain comprehensive knowledge about landslides while personalizing the prediction model to predict landslides accurately in the target region. *SlideSafe* utilizes contrastive learning in the feature extraction phase to derive general knowledge about landslides, as it does not rely on label information. This enables us to obtain meaningful representation without feeding each component's class imbalance ratio, which sets it apart from existing methods for class-imbalanced federated learning [30]. Furthermore, *SlideSafe* can enhance the efficiency of the learning process for each client and personalize each model based on the similarity assessment of clients and corresponding grouping.

A. Virtual Gridding Module

To make natural phenomena that are continuously iterable by the machine learning model, this module is responsible for transforming real-world data into a square grid format for data discretization. *SlideSafe*'s default cell edge length is 1km, and all data is represented in this square-grid cell format.

B. Feature Extractor Module

1) Multi-View Encoder: Rainfall-induced landslide depends on rainfall and the land characteristics (*i.e.*, soil, plant, slope, and elevation). To capture the effects of these factors, we employ a multi-view encoder that receives the time series data (*i.e.*, rainfall) and the land characteristics. Since underground water flow plays a significant role in landslides, it is essential to consider the target area and its nearby surroundings to

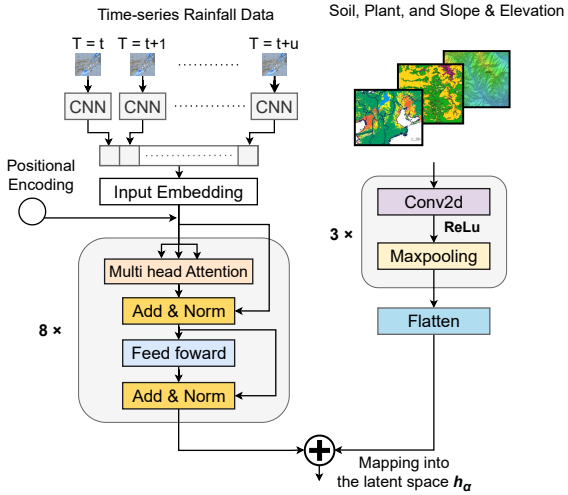


Fig. 2. Multi-input Feature Encoder. We employ eight transformer layers and three CNN layers.

accurately predict landslides. As shown in Figure 2, we extract the feature from a sequence of gridding data using ConvTransformer [31] and extract the feature from land characteristics data using CNN.

2) Feature Extractor with Contrastive Learning: This part obtains meaningful representation from the spatio-temporal inputs in grid-structured data. To avoid the effects of class imbalance and get feasible representation from complex spatio-temporal features, we incorporate a contrastive learning mechanism. Inspired by recent contrastive learning such as SimCLR [20] in the computer vision research domain, our feature extractor learns representations by maximizing agreement between differently augmented views of the same data example via a contrastive loss in the latent space. This framework comprises the following four major components. **Stochastic data augmentation**: From each data sample, it randomly generates two correlated views of the sample, denoted as $\tilde{x}_\alpha$ and $\tilde{x}_\beta$. They are considered a positive pair. In *SlideSafe*, we sequentially apply two simple augmentations: random cropping followed by resizing back to the original size, and

TABLE I
EXPERIMENTAL SETTINGS

Parameter	Explored range (**Bold**: default value)
Cell edge length [km]	{**1**, 2}
Time interval [hour]	{**6**, 12}
Batch size	{8, 16 , 32, 64, **128**}
Temperature τ	{0.3, **0.5**, 0.8}
Initial communication round T_{init}	{10, 20, 50, **100**}
Communication round T	{10, 20, **30**}
Local epoch E	{10, **30**, 50}
Similarity threshold m	{-0.1, 0, 0.1, **0.3**, 0.6, 1.0}
Parameter of similarity metrics γ	{0.2, **0.4**, 0.6, 0.8}
Model size of *SlideSafe*	{0.76 Million}

random rotation to obtain $\tilde{x}_\alpha$ and $\tilde{x}_\beta$. **Encoder** $f(\cdot)$: It extracts representation vectors from the data augmented in the previous component. We employ an encoder that consists of two types of view to obtain $h_\alpha = f(\tilde{x}_\alpha)$ where $h_\alpha \in \mathbb{R}^d$ is a d-dimensional output of the multi-view encoder. **Small neural network projection head** $g(\cdot)$: It maps representations to the space where contrastive loss is applied. We use a shallow neural network with one hidden layer to obtain $z_\alpha = g(h_\alpha)$. **Contrastive loss prediction**: A contrastive loss function is defined for this contrastive prediction task, given a set $\{\tilde{x}_k\}$ including a positive pair $\tilde{x}_\alpha$ and $\tilde{x}_\beta$ of samples.

We randomly pick up B examples for a minibatch and generate $2B$ augmented data using stochastic data augmentation. Then, we defined the contrastive prediction task on the augmented data, where for each positive pair, we applied contrastive learning regarding the rest of the $2(B-1)$ augmented examples as negative examples. Let $\text{sim}(u, v) = u^\top v/||u||\cdot||v||$ denote the dot product between two normalized u and v (i.e. cosine similarity). Then the loss function for a positive pair of examples $(\tilde{x}_\alpha, \tilde{x}_\beta)$ is defined as:

$$\ell_{\alpha,\beta} = -\log \frac{\exp(\text{sim}(z_\alpha, z_\beta)/\tau)}{\sum_{k \in 2B\_index\_set} b_{[k \neq \alpha]} \exp(\text{sim}(z_k, z_\beta)/\tau)} \tag{3}$$

where the value of $b_{[k\neq\alpha]} \in \{0, 1\}$ is 1 if and only if $k \neq \alpha$, and τ denotes a temperature parameter. The final loss is computed across all positive pairs, both $(\tilde{x}_\alpha, \tilde{x}_\beta)$ and $(\tilde{x}_\beta, \tilde{x}_\alpha)$ in a mini-batch. This loss has been used in [20] and called *NT-Xent loss (the normalized temperature-scaled cross-entropy loss)*.

C. Decentralized Collaborate Learning Mechanism

This section presents a decentralized collaborative learning approach that ensures performance for each client. In general, model performance can be affected when aggregating models trained on dissimilar data distributions. To achieve high model performance even in non-IID settings, *SlideSafe* incorporates a similar neighbor selection mechanism [28]. This mechanism aims to identify clients whose data distribution is similar to each other, enhancing the performance of each model. It is called the personalized knowledge-sharing mechanism in *SlideSafe*, consisting of two stages: *similar client selection* and *distributed model update*.

1) Similar Client Selection: In this stage, the global server of *SlideSafe* aims to identify similar client for each local client (i.e. municipalities). Each local client is denoted as i and possesses a model with model parameters w_i. These parameters are updated over E local epochs and subsequently uploaded to a *global server* for a total of T_{init} rounds. This procedure mirrors the commonly used approach in centralized federated learning.

We measure the similarity of clients using the Personalized Adaptive Neighbor Matching (PANMGrad) [28] method, which does not require setting the number of clusters. The global server calculates the similarity of models of two clients i and j using a combination of two parameters. The first parameter is given in the first term of Eq. (4), where $\gamma \in [0, 1]$ is a hyperparameter, and g_i^t is the vectorized gradient of client i in round t and is expected to represent the data distribution in client i. g_i^t is obtained by $g_i^t = w_i^t - w_i^{t-1}$, where t is the current time round. We note that w_i^t is initialized by the same global model at the beginning of local training in each round, and one-round update g_i^t can be noisy. To capture the historical optimization directions of each model, we also introduce accumulated weight updates from the initial model, $h_i^t = w_i^t - w_i^0$, and use it in the second parameter given by the second term of Eq. (4). h_i^t is the accumulated vectorized gradient of client i in round t. The global server calculates the similarity of models of two clients using similarity metrics given in Eq. (4).

$$\text{sim}_{i,j} = \gamma \cdot \frac{< g_i^t, g_j^t >}{||g_i^t|| \cdot ||g_j^t||} + (1-\gamma) \cdot \frac{< h_i^t, h_j^t >}{||h_i^t|| \cdot ||h_j^t||} \tag{4}$$

The global server selects the clients whose similarity is higher than the threshold m as the "neighbor clients" using average similarity during T_{init}. The threshold m is the system parameter.

2) Distributed Model Update: In this stage, each client communicates with each of the similar clients found in the previous stage. More concretely, the model parameters w_i of client i are delivered to each client j which is a similar client of i. Client i then waits for the models sent from up to N_{sim_i} clients and aggregates these received models. We use n to represent the number of received models, and it is important to note that $n \leq N_{\text{sim}_i}$, taking into account cases where model delivery may fail or not be performed due to some reasons. Eq. (5) is the federated aggregation of the received models.

$$\bar{w}_i^{t+1} \leftarrow \frac{1}{n+1} \sum_{j \in \{\text{similar clients of } i\} \cup \{i\}} w_j^t \tag{5}$$

The new aggregated model $\bar{w}_i^{t+1}$ is then trained for E epochs before it is ready to be gossiped again.

D. Client Tailored Classification

This module takes the output (*i.e.*, latent features) of the feature extractor module as input, and classifies the given inputs into positive and negative cases, which is output in confidence using the Sigmoid function. It is two layers shallow neural network and trained by a cost-sensitive loss function

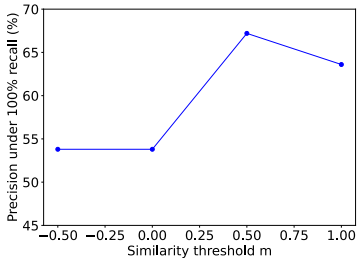


Fig. 3. Impact of Similarity Threshold m

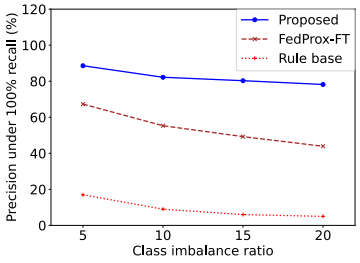


Fig. 4. Impact of Class Imbalance Ratio

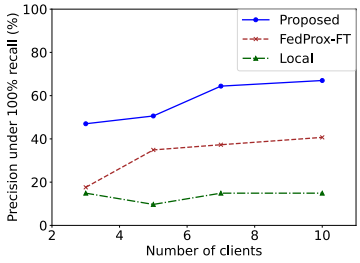


Fig. 5. Impact of Number of Clients

$\ell(y,\hat{y}) = -r \times y \log(\hat{y}) - (1-y)\log(1-\hat{y})$. The loss function calculates the loss according to the ratio of the classes, where y and $\hat{y}$ are the actual label and output of the model respectively, and r is the ratio of the negative case and positive case.

Since landslide events are rare and geologically skewed, some clients may have never encountered landslides. In such situations, fine-tuning those clients may not work as they do not hold positive case data. To enable prediction for such clients (called *unlabeled clients*), *SlideSafe* provides the model among pre-trained models to those clients. Each unlabeled client, say u, can train its multi-view encoder and obtain model parameters w_u using $z_u \sim D_u$ through contrastive learning because contrastive learning does not rely on label information. Since the multi-view encoder is trained to obtain high-quality latent representation from rainfall and terrain data, the multi-view encoder's weight is expected to reflect rainfall and terrain patterns in this region. Thus, *SlideSafe* can select a similar client even for unlabeled clients. The global server selects the most similar model by comparing w_u with each model and finds the most similar client. Finally, unlabeled client u receives the model of the most similar client and applies it to its data to predict landslide occurrence.

V. Evaluation

A. Data Collection and Configuration

We have collected data related to landslide prediction for ten prefectures in Japan. Landslides depend on various features, and we have curated datasets with well-established and commonly used variables. **Landslide Events:** We collected records of landslide events in Japan from 2021 to 2022, totaling 253 events. Each landslide event is documented with its corresponding time and location, represented as six decimal points for latitude and longitude coordinates. The time granularity in our dataset is at the day level, and *SlideSafe* predicts landslide occurrences at this level, which is sufficient for pre-evacuation planning. **Rainfall:** As demonstrated in [4], time series data of rainfall contribute to accurate landslide prediction because water accumulation plays a crucial role in landslides. Our dataset contains 2 years of data from January 1st, 2021 to December 31, 2022. **Soil Property:** Soil water content is a significant factor contributing to landslides because underground water flow depends on soil properties. The properties related to the ease of sliding or water drainage have a significant impact on landslide events. In our dataset, the soil is categorized into 10 classes, including artificial paved areas. **Vegetation:** The presence of vegetation in the surrounding area is a significant factor contributing to landslides, as plant and tree roots help stabilize underground water and soil. Our data sets include 11 types of vegetation. **Elevation and Slope Angle:** Elevation and slope angle are crucial factors in landslide prediction because they are closely linked to water accumulation and the gravitational force acting on the land. Additionally, steeper slopes are inherently more susceptible to landslides than gentler ones. Elevation and slope are represented as real numbers with eight categorized directions in our datasets.

We adhere to the Japan MESH3 Boundaries ECM (meshing system in Japan) and collect the data for each mesh as a unit, where the mesh length is one kilometer. To train and evaluate our system, we have prepared negative cases, representing days when no landslides occurred. For each client, we collected data for the days when current warning information was issued to ensure the quality of these negative cases. Additionally, we randomly selected an equal number of data points for each client to create negative cases. Furthermore, we show the experimental configuration settings in Table I.

B. Evaluation Metrics

The primary objective of *SlideSafe* is to achieve accurate landslide prediction. In disaster scenarios, achieving a recall rate of 100% is crucial because this information serves as the vital trigger for evacuations. However, it is also important to maintain a low false positive rate, as an excessive number of false alarms can erode trust in the information provided. To evaluate the practical utility of disaster information, we employ a performance metric known as *precision under 100% recall*. This metric allows us to assess the model's precision while ensuring that recall remains at 100%. Given that *SlideSafe* aims to predict landslides in distributed settings, all reported performance metrics are averaged across clients for each distribution, providing a comprehensive assessment of the system's effectiveness.

C. Decentralized Prediction Model Evaluation

In this section, to measure the performance of *SlideSafe*, we compare the proposed system with the most relevant state-of-the-art techniques: FedAvg [32], FedProx [22], and FedProx with fine-tuning (FedProx-FT) [25]. FedAvg [32] is the most classic federated learning method, which is the baseline of the training results. FedProx [22] is a widely used method to make the training process efficient in non-IID datasets. FedProx uses a regulation parameter μ to prevent the local

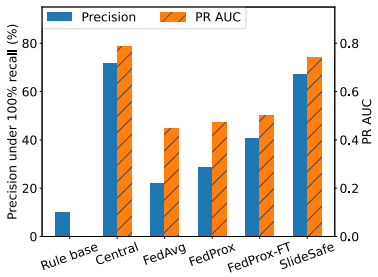


Fig. 6. Precision under 100% recall and precision and recall AUC (PR AUC)

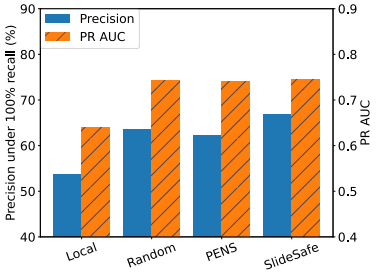


Fig. 7. Impact of client selection algorithm

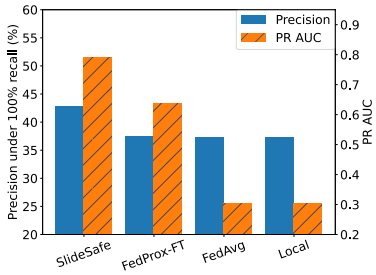


Fig. 8. Prediction performance in unseen region

model from deviating greatly from the global model. We fix the regulation parameter μ to 0.3. FedProx-FT [25] aims to build personalized models in non-IID settings by fine-tuning the global model trained in FedProx using local data. Furthermore, we set current Japanese warning information (Rule base) operated in the real world as a baseline.

1) Impact of Similarity Threshold: In *SlideSafe*, for each client, similar clients are selected to share the trained models, and this selection is controlled by m, the similarity threshold parameter. A smaller value of m results in grouping dissimilar clients and an increase in the impact of non-IID data distribution. Meanwhile, with a large value of m, the precision may also decrease because clients cannot collect models from sufficient clients to obtain comprehensive knowledge about landslides in decentralized collaborative learning. Therefore, we study the effect of m and consider the appropriate value through the experiment. Figure.3 shows the relationship between the threshold and the precision under 100% recall. We can observe that *SlideSafe* performs best with $m = 0.5$.

2) Robustness: SlideSafe aims to predict landslides in the real world. Such systems must be robust to tough situations because the settings and data are not necessarily ideal. We evaluate *SlideSafe*'s robustness from the number of clients. Generally, federated learning performs better as the number of clients increases. However, traditional federated learning methods struggle with non-IID data. Figure.5 presents the precision rates with different numbers of clients. Notably, we observe that *SlideSafe* and FedProx-FT [25] exhibit increasing performance trends as more clients are involved. It is worth noting that *SlideSafe* achieves higher performance with just three clients than FedProx-FT [25] does with ten clients. This highlights the efficacy of *SlideSafe*, stemming from its mechanism for grouping similar clients in federated learning.

The class imbalance ratio of our target dataset differs from clients (regions). Therefore, we observe the impact of class imbalance ratios on the precision. Figure 4 shows the results. Although all the methods are affected by the increase of the ratio, the decreasing trend of *SlideSafe* is much slower than the other two comparisons and the precision rates are much higher than those of the others. This indicates that *SlideSafe* is robust to class imbalance.

3) Performance Comparison: Figure 6 displays the precision under 100% recall and PR AUC of each method. In our experiments, we have ten local clients, and the values of other relevant parameters can be found in Table I. *SlideSafe* takes 1.5 seconds to train the model for each epoch and 300ms to inference on average. Our machine environment is the following: GPU is NVIDIA GeForce RTX 4090, CPU is AMD Ryzen 9 7950X 16-Core Processor, and RAM is 64GB. Since *SlideSafe* works on local municipalities and needs only 300ms in runtime, local municipalities can make predictions in real time. It is worth noting that the centralized model in the graph utilizes data from all clients, making its performance a reference for an ideal and best-case scenario. In our observations, *SlideSafe* achieves the highest performance among the other state-of-the-art methods within decentralized settings. This outcome highlights that *SlideSafe* excels by personalizing models for target regions while acquiring comprehensive knowledge through decentralized collaborative learning with similar regions.

SlideSafe utilizes Personalized Adaptive Neighbor Matching (PANMGrad) [28] to identify clients with similar data distributions. To assess the impact of this approach, we compare *SlideSafe* with two others: random gossip (Random) and Performance-based node selection (PENS) [6]. In the random method, clients communicate with each other randomly, potentially leading to a mix of dissimilar data distributions. PENS is a state-of-the-art method for finding clients with similar data distributions. It selects similar clients by leveraging the loss values of client j's model validated on client i's local datasets. While PENS is a promising metric, it requires $O(N^2)$ communications to identify similar clients, where N is the number of clients in the network. Figure 7 presents precision under 100% recall and PR AUC for each method. Notably, *SlideSafe* outperforms other methods in terms of precision. This superior performance is attributed to *SlideSafe*'s successful formation of client groups with similar data distributions, enabling efficient knowledge sharing. It should also be highlighted that *SlideSafe* requires fewer communications, specifically $O(N)$ communications.

Finally, to assess the performance of *SlideSafe* in regions that have never experienced landslides, we examine its performance in unseen regions by applying a trained model to these unfamiliar areas. We select the model to use in these unseen regions using the PANMGrad approach. In *SlideSafe*, clients can identify similar clients without relying on labels, allowing us to apply the model in the region most similar to the target region. Figure 8 shows the model's performance in un-

familiar regions, which demonstrates *SlideSafe* predicts more accurately in unfamiliar regions than the existing methods.

VI. Conclusion

This paper presents a novel landslide prediction system combining spatio-temporal contrastive learning and selective collaborative learning. It begins by training a contrastive learning model to extract meaningful representations of land characteristics in each region. Subsequently, these trained models are merged among only regions with similar characteristics, leveraging collaborative learning. The federated models are then fine-tuned and customized for the landslide event prediction using the data specific to each region. We experimented with real-world datasets from ten Japanese prefectures. The results of these experiments illustrate that our approach achieves the highest precision when aiming for a 100% recall rate in landslide prediction, surpassing the performance of state-of-the-art methods. These findings underscore that our system performs well in alerting to signs of landslides while maintaining a low false-positive rate. In the future, we plan to boost the prediction reliability with explainable AI techniques (e.g., SHAP and LIME) and apply the framework to other types of disasters, such as floods and earthquakes.

Acknowledgment

This work was partially funded by JST, CREST Grant JPMJCR21M5, JSPS, KAKENHI Grant 22K12011 and 23H03384, and NVIDIA award. The authors appreciate JMA, MLIT, JMBSC, MOE, and GSI of Japan, which provide datasets to the authors. The JMA data used in this study was provided by way of the "Meteorological Research Consortium", a framework for research cooperation of JMA and MSJ.

References

[1] H. Rizk, Y. Nishimur, H. Yamaguchi, and T. Higashino, "Drone-based water level detection in flood disasters," *International journal of environmental research and public health*, vol. 19, no. 1, p. 237, 2021.

[2] T. S. Teja, A. Dikshit, and N. Satyam, "Determination of rainfall thresholds for landslide prediction using an algorithm-based approach: Case study in the darjeeling himalayas, india," *Geosciences*, vol. 9, no. 7, 2019.

[3] E. Collini, L. I. Palesi, P. Nesi, G. Pantaleo, N. Nocentini, and A. Rosi, "Predicting and understanding landslide events with explainable ai," *IEEE Access*, vol. 10, pp. 31 175–31 189, 2022.

[4] P. Xie, A. Zhou, and B. Chai, "The application of long short-term memory (lstm) method on displacement prediction of multifactor-induced landslides," *IEEE Access*, vol. 7, pp. 54 305–54 311, 2019.

[5] T. A. Steelman, S. M. McCaffrey, A.-L. K. Velez, and J. A. Briefel, "What information do people use, trust, and find useful during a disaster? evidence from five large wildfires," *Natural Hazards*, vol. 76, 2015.

[6] N. Onoszko, G. Karlsson, O. Mogren, and E. L. Zec, "Decentralized federated learning of deep neural networks on non-iid data," 2021.

[7] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 10 713–10 722.

[8] Z. Liu, D. Guo, S. Lacasse, J.-h. Li, B. Yang, and J. C. Choi, "Algorithms for intelligent prediction of landslide displacements," 2020.

[9] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019.

[10] E. Tasci, Y. Zhuge, K. Camphausen, and A. V. Krauze, "Bias and class imbalance in oncologic data—towards inclusive and transferrable ai in large scale oncology data sets," *Cancers*, vol. 14, no. 12, p. 2897, 2022.

[11] A. A. Alsaui, Y. A. Alghofaili, M. Alghadeer, and F. H. Alharbi, "Resampling techniques for materials informatics: limitations in crystal point groups classification," *Journal of Chemical Information and Modeling*, vol. 62, no. 15, pp. 3514–3523, 2022.

[12] A. Singh, R. K. Ranjan, and A. Tiwari, "Credit card fraud detection under extreme imbalanced data: a comparative study of data-level algorithms," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 34, no. 4, pp. 571–598, 2022.

[13] B. Krawczyk, C. Bellinger, R. Corizzo, and N. Japkowicz, "Undersampling with support vectors for multi-class imbalanced data classification," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–7.

[14] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[15] D. Dablain, B. Krawczyk, and N. V. Chawla, "Deepsmote: Fusing deep learning and smote for imbalanced data," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[16] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, 2018, pp. 117–122.

[17] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[18] F. Feng, K.-C. Li, J. Shen, Q. Zhou, and X. Yang, "Using cost-sensitive learning and feature selection algorithms to improve the performance of imbalanced classification," *IEEE Access*, vol. 8, 2020.

[19] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 4368–4374.

[20] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[21] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *Journal of Parallel and Distributed Computing*, vol. 148, 2021.

[22] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 429–450.

[23] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient federated learning with sketching," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8253–8265.

[24] N. Dong and I. Voiculescu, "Federated contrastive learning for decentralized unlabeled medical images," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2021, pp. 378–387.

[25] K. Wang, R. Mathews, C. Kiddon, H. Eichner, F. Beaufays, and D. Ramage, "Federated evaluation of on-device personalization," 2019.

[26] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang, "Personalized cross-silo federated learning on non-iid data," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 9, 2021.

[27] R. Dai, L. Shen, F. He, X. Tian, and D. Tao, "Dispfl: Towards communication-efficient personalized federated learning via decentralized sparse training," *arXiv preprint arXiv:2206.00187*, 2022.

[28] Z. Li, J. Lu, S. Luo, D. Zhu, Y. Shao, Y. Li, Z. Zhang, Y. Wang, and C. Wu, "Towards effective clustered federated learning: A peer-to-peer framework with adaptive neighbor matching," *IEEE Transactions on Big Data*, pp. 1–16, 2022.

[29] A. Amini, M. Dolatshahi, and R. Kerachian, "Adaptive precipitation nowcasting using deep learning and ensemble modeling," *Journal of Hydrology*, vol. 612, p. 128197, 2022.

[30] L. Wang, S. Xu, X. Wang, and Q. Zhu, "Addressing class imbalance in federated learning," 2020.

[31] Z. Liu, S. Luo, W. Li, J. Lu, Y. Wu, S. Sun, C. Li, and L. Yang, "Convtransformer: A convolutional transformer network for video frame synthesis," 2021.

[32] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017.

Reinforcement Learning Applicability for Resource-Based Auto-scaling in Serverless Edge Applications

Priscilla Benedetti
Univ. of Perugia, Dept. of Eng.
via G.Duranti 93, Perugia, Italy
Vrije Universiteit Brussel, ETRO
Pleinlaan 2, Brussels, Belgium
priscilla.benedetti@vub.be

M. Femminella, G. Reali
Univ. of Perugia, Dept. of Eng, CNIT RU
via G.Duranti 93, Perugia, Italy
name.subname@unipg.it

Kris Steenhaut
Vrije Universiteit Brussel
ETRO Dept.
Pleinlaan 2, Brussels, Belgium
ksteenha@etrovub.be

***Abstract*—Serverless computing is an alternative deployment paradigm for cloud computing platforms, aimed to provide scalability and cost reduction without requiring any additional deployment overhead from developers. Generally, open-source serverless computing platforms rely on two auto-scaling approaches: workload-based and resource-based. In the former, a designated algorithm scales instances according to the number of incoming requests. In the latter, instances are scaled when a certain resource usage limit, such as maximum Central Processing Unit (CPU) utilization, is reached. Resource-based auto-scaling is usually implemented leveraging Kubernetes Horizontal Pod Autoscaler (HPA). In this work, we investigate the applicability of a reinforcement-based approach to resource-based auto-scaling in OpenFaaS, the most widely used open-source serverless platform. Serverless technologies are particularly convenient when dealing with edge computing on constrained devices or resource-limited machines. Our experimental analysis has been conducted on constrained Kubernetes-based nodes, to simulate such an edge application scenario. Its preliminary results show that our proposed model learns an effective scaling policy, based on CPU utilization, to provide minimal service latency within a limited number of iterations.**

***Index Terms*—edge computing, serverless computing, reinforcement learning, Kubernetes, OpenFaaS**

I. Introduction

Serverless computing has been introduced to enhance multiplexing and scalability functions in computing platforms [1]. Whereas traditional cloud computing frameworks allocate compute resources in advance, serverless technologies make use of dynamic resource allocation, through an event-driven allocation control, allowing the number of service instances to be zero when there is no demand while scaling to as many instances as needed by the incoming requests, leading to reduced cost. Serverless computing, usually implemented using Functions-as-a-Service (FaaS), only focuses on the application code, organized through individual services, typically event-triggered and executed in containers. Infrastructure provisioning and maintenance are handled by the serverless platform and are completely hidden from developers and users [2]. Instead of deploying a whole platform or an application in the cloud servers, by using FaaS just functions are required, as components of complex applications. Such functions can be loaded as virtual containers when needed, on demand, and possibly in parallel, without any need for controlling application deployment processes at operating system level. This means that containers are dynamically scheduled in the hardware infrastructure maintained by cloud providers. This approach, focused on dynamic provisioning and resource efficiency, makes serverless computing a promising deployment model for edge platforms [3] [4]. Through the use of advanced virtualization technologies, FaaS can ensure that the volume of resources consumed by an application is dynamically controlled and is tailored to the actual computing needs.

This auto-scaling capability is handled with different mechanism within the available serverless solutions. Auto-scaling can be either workload-based, *i.e.*, providing additional resources when incoming traffic increases as done with Amazon Web Services (AWS) Lambda, or it can be resource-based [5], using the resource-based Kubernetes Horizontal Pod Autoscaler (HPA) [6] to trigger scaling via per-instance CPU or memory utilization thresholds. In this work, we focus on resource-based auto-scaling. Resource-based auto-scaling is available in all Kubernetes-based serverless platforms, *i.e.*, the majority of the available open source serverless solutions [7]. Without loss of generality, we decided to deploy and analyse the feasibility of a reinforcement learning (RL) model to dynamically set the optimal configuration for resource-based auto-scaling in OpenFaaS [8], the open-source FaaS solution with highest adoption rate [7]. RL implies the development of an agent learning an optimal policy through a certain number of trial-and-error interactions with the environment. Since this approach has been successful in the field of virtual machines auto-scaling, there is a growing interest in the applicability of RL to serverless platforms auto-scaling mechanisms [9].

Therefore, in this work we present some preliminary results on RL for resource-based auto-scaling optimization in serverless platforms. Specifically, we implemented a simple serverless application in an edge computing use case scenario,

978-1-6654-1647-4/22/$31.00 ©2022 IEEE

relying on nodes with limited resources. On this testbed, two consecutive experiments are conducted. First, a baseline analysis of application latency variation under different CPU-consumption auto-scaling configurations is done. It demonstrates the dependency of application latency on the specific auto-scaling settings. Subsequently, given the baseline results, we evaluate the ability of a RL algorithm for optimal resource-based auto-scaling configuration in OpenFaaS. Results show that the developed agent can learn within a reasonable time an effective scaling configuration policy, which provides a lower response time than the one obtained with the serverless platform's default CPU-consumption auto-scaling settings.

The rest of the paper is organized as follows. Section II introduces OpenFaaS and reinforcement learning theory, specifically Q-learning. Section III presents related work in serverless platforms and general cloud-computing auto-scaling techniques. Section IV provides an overview of the experimental setup of this work, describing the testbed in use. Section V illustrates the impact of different CPU-rate auto-scaling configurations on application latency, while Section VI delves into the RL agent development and evaluation. Section VII concludes the paper with further comments, remarks on limitations and possible future work.

II. Background

In this section we provide a brief introduction of the OpenFaaS serverless platform, with a particular focus on resource-based auto-scaling via Kubernetes HPA. Moreover, we give an overview of Q-learning, the reinforcement learning algorithm used in this work.

A. OpenFaaS

OpenFaaS [8] is a cloud native computing foundation (CNCF) open source serverless platform. It allows developers to define and use templates of different languages to create and build serverless functions. It relies on Docker images and Kubernetes control plane to run applications, providing failover, high availability, scale-out and secret management. As shown in Fig.1, the OpenFaaS gateway, which is similar to a reverse proxy, is in charge of exposing and managing the function Pods, offering a Representational State Transfer (REST) Application Programming Interface (API) for all interactions. For workload-based auto-scaling, OpenFaaS is set by default with a single auto-scaling rule defined for the *AlertManager* component. The *AlertManager* will monitor requests-per-second metrics provided by a *Prometheus* [10] instance in the cluster and trigger auto-scaling when needed. Conversely, when a resource-based auto-scaling approach is considered, OpenFaaS does not rely on *AlertManager* and *Prometheus* components. The auto-scaling process will be completely governed by Kubernetes Horizontal Pod Autoscaler (HPA), which supports CPU utilization metrics as well as application-provided metrics and custom metrics. As shown in Fig.1, the HPA controller gets resource usage metrics from the API provided by Kubernetes native *metrics-server*. Then, if a target utilization value is set, the HPA controller calculates

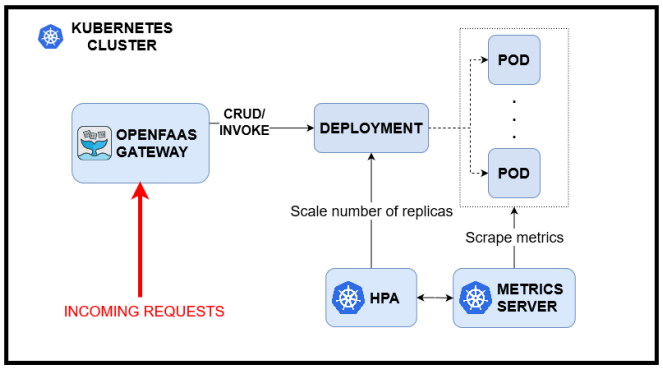


Fig. 1: Conceptual workflow of OpenFaas resource-based auto-scaling using Kubernetes HPA.

the utilization value as a percentage of the equivalent resource request on the containers in each Pod and produces a ratio used to scale the number of desired replicas. HPA operates on the number of Pods by using Deployment's Scale, which is an interface that allows to dynamically set the number of replicas and examine their current state [6].

B. Q-learning

Reinforcement learning problems involve mapping situations to actions so as to maximize a numerical reward signal. In particular, Q-learning is one of the most applied representative reinforcement learning approaches, which uses an off-policy control that separates the deferral policy from the learning policy and updates the action selection using the Bellman optimal equations and the ϵ-greed policy. Q-learning is an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states [11] [12]. Q-learning is aimed at training in a stepwise fashion an approximator $Q_\theta(s,a)$ of the optimal action-value function Q^*. $Q_\theta(s,a)$ identifies the cumulative reward expected by the agent when taking an action a in a state s. Considering the actual reward at each iteration, the optimization of Q is performed incrementally per step t,as shown below in formula 1:

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t)+\alpha[r_t+\gamma max_a Q(s_{t+1},a)] \quad (1)$$

α is the learning rate, r_t is the immediate reward received when moving from s_t to s_{t+1} and γ a discount factor to balance between immediate and future rewards. During the training phase, the agent has the ability to choose between trying an action in a random way (exploration) or selecting the current best action (exploitation), *i.e.*, the action with the highest Q-value. A common approach is storing the Q-value for each state-action in a lookup table, *i.e.*, the Q-table, which serves as a basis for the agent's decision-making in the exploitation case. The probability of exploration versus exploitation is defined by an ϵ coefficient, which defines the probability of exploration and usually decreases as the training process advances.

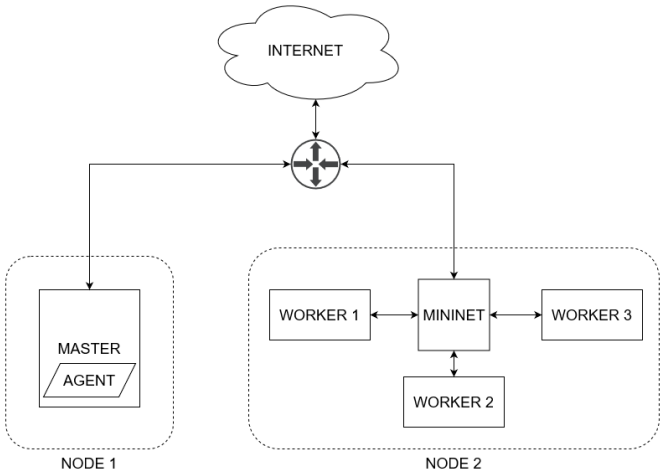


Fig. 2: Testbed setup for the experiments.

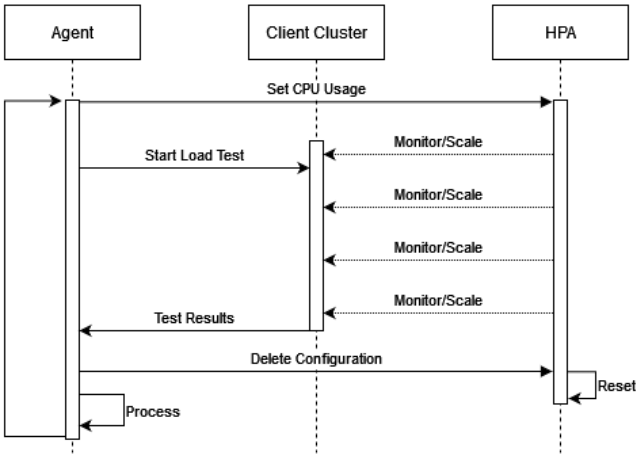


Fig. 3: RL agent training process flow for each iteration.

III. Related Work

Serverless computing feasibility for edge platforms has been introduced in [1], where the authors present a general description of serverless computing and its relevant programming model, pointing to Function-as-a-Service (FaaS) platforms as the concretisation of serverless computing principles. Moreover, a serverless edge platform based on OpenWhisk is presented in [13], along with different application scenarios and an evaluation in terms of memory footprint, latency, throughput, and scalability. Another example of serverless application in an edge computing scenario is provided in [14], where the authors rely on OpenFaaS to deploy a monitoring application for IoT data deployed on low-hardware nodes to simulate an edge layer. There is a vast offering in open-source serverless computing platforms. Thus, some studies can be found that compare these solutions from different points of view, taking in account also scalability features. In [15], an assessment of serverless platforms is conducted on top of a Kubernetes cluster. [7] and [16] perform tests for measuring some interesting metrics such as the response time, the ratio of successful responses or the impact of auto-scaling on some of these metrics. The study on open-source serverless platforms presented in [5] includes a detailed comparison of auto-scaling performance, taking in account both resource-based and workload-based policies. Nevertheless, these works do not consider possible alternatives such as RL to the default auto-scaling mechanisms present in the analyzed platforms. In fact, reinforcement learning has been adopted by many researchers for organising scaling in the field of VM provisioning and de-provisioning, such as in [17] and [18]. Moreover, auto-scaling configuration for container-based applications has been widely explored [19] [20]. Regarding the applicability of RL-based algorithms to optimize serverless platform auto-scaling, [9] presents an RL-based model to optimize requests-per-second (RPS) throughput tuning serverless auto-scaling mechanisms. The research considers workload-based auto-scaling for Knative, which is an open-source serverless platform based on Kubernetes. As far as we know, the use of RL for optimizing resource-based auto-scaling in serverless platforms has not been analysed at present.

IV. Testbed Setup

The testbed architecture for our experiments is shown in Fig.2. To simulate an edge computing layer, we deployed a Kubernetes cluster on 4 virtual machines, hosted on two physical nodes residing in the same Local Area Network (LAN). Node 1 is equipped with a 4-core CPU and 8GB RAM, Node 2 with a 12-core CPU and 32 GB RAM. Each virtual machine, simulating a node of the edge cluster, was equipped with 2 vCPU and 4 GB of memory. The three workers are in communication via Mininet[1], a virtualized network emulator. The OpenFaaS serverless platform is hosted on the worker nodes, as well as the Kubernetes *metrics-server* needed by the Horizontal Pod Autoscaler. The master is including the reinforcement-learning agent developed for this work. The serverless application deployed on this platform is a web-server showing an HTML text, *show-html*, developed using OpenFaaS Python Flask template[2], which relies on *of-watchdog*, OpenFaas' reverse proxy for HTTP microservices. To first analyse the OpenFaaS resource-based auto-scaling process and then develop the RL-based optimization, we considered HPA auto-scaling based on the average CPU utilization across all serverless application's Pods, setting the minimum number of replicas to 1 and the maximum to 200. With the first baseline experiment, we analysed the throughput, in terms of execution time, of *show-html*, considering every HPA's CPU usage percentage setting in the interval [10,...90]. For each HPA configuration, the throughput has been evaluated under a load test with an increasing number of incoming parallel requests, for a load test duration of 5 minutes. To simulate

[1]http://mininet.org/

[2]https://github.com/openfaas/python-flask-template

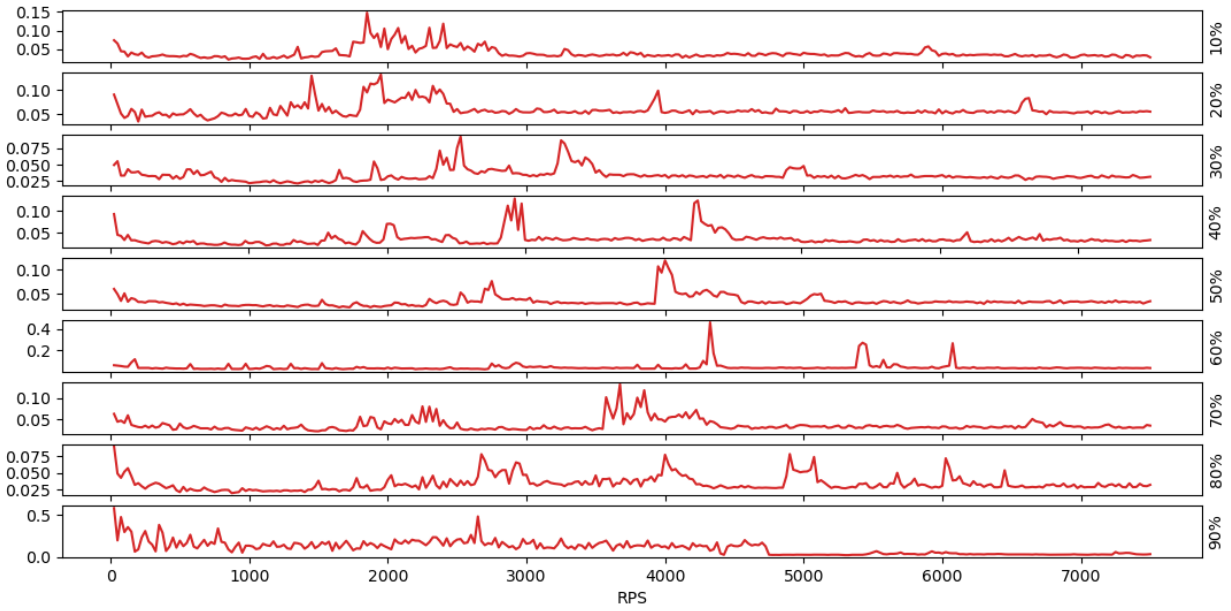


Fig. 4: *show-html* latency, [s], for each CPU usage setting considered during the load test.

the load test, we relied on the heavy-duty Hey HTTP load generator[3], considering 25 parallel requests senders.

For the Q-learning experiment, the process flow of one training iteration is shown in Fig.3. At each iteration, the agent sets the HPA according to its CPU usage percentage choice. Then, the same load test of the baseline experiment is executed, while HPA, communicating with the Client Cluster *metrics-server*, scales replicas to maintain serverless application Pods CPU usage below the configured threshold if needed. When the test ends, the agent scrapes the latency results, deletes the current HPA configuration and processes its decision for the next iteration. It must be noted that the agent waits for Pods replicas to come back to their initial minimum number before proceeding to the next iteration.

V. Baseline Experiment

During the baseline experiment, a load test has been executed for each CPU usage percentage considered, to assess the best configuration. The load test, performed via Hey generator, sent an incremental number of requests from 25 concurrent entities to the OpenFaaS gateway, to call *show-html* serverless function, up to 7500 requests per seconds (RPS). We considered the report produced by Hey to get the average latency of the responses per second, considering 300 s with an increasing number of requests per second from 0 to 7500. Before each load test, *show-html* replicas are set back to the minimum. Fig.4 depicts average latency evolution during Hey load test for each CPU usage setting considered. The presence of latency spikes can be observed in every test, this phenomenon is due to the auto-scaling process. Indeed, the waiting time for the new replicas to be up and running increases the overall execution time for the considered serverless function.

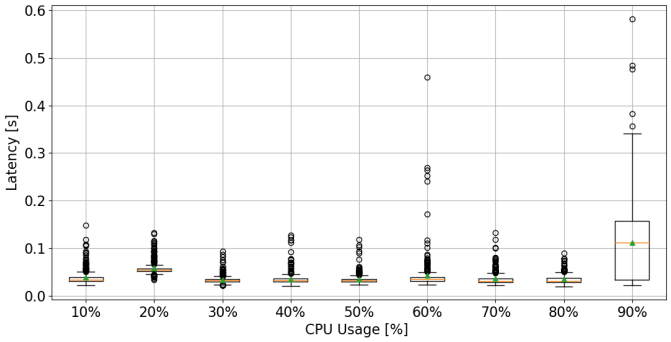


Fig. 5: Boxplots of the average latency for each CPU usage setting. The overall best result is obtained for 30%: 0.0341653 s.

To have a better insight of latency variation with different configurations, the boxplots of average latency for each CPU setting over the load test is shown in Fig. 5. The mean is shown as a green triangle. From this plot, there is an improvement of latency in the interval from 30% to 50% CPU usage, with good performance for the 80% as well. The overall best performance is achieved for the 30% CPU usage setting, leading to the minimal latency, *i.e.*, 0.0341653 s. A manifest performance degradation for the 90% CPU usage setting can be observed. This is due to several instabilities in the latency results, as can be seen in the previous figure (Fig.4). Forcing the cluster to not scale replicas until the CPU is almost saturated leads to a pronounced latency increase and many Pods crashes. It must be noted that the monitoring components as well have a CPU footprint which influences the overall auto-scaling behaviour,

[3]https://github.com/rakyll/hey

as previously pointed out in [5]. This issue will be further commented in Section VII.

VI. Q-learning Experiment

The previously described baseline experiment showed the impact of CPU usage auto-scaling settings on latency. Hence, in this second experiment we evaluate the applicability of Q-learning models to learn effective resource-based scaling policies. Instead of incrementally increasing maximum CPU usage allowed for serverless Pods, the agent uses knowledge of the environment, encoded using states, to test different CPU usage auto-scaling settings (actions), evaluating them by obtained rewards. The states, which provide all relevant information about system's conditions, can include a large number of features influencing the throughput, such as network utilization or memory usage. For this preliminary analysis, we limited the considered states to CPU usage settings in use at each iteration. This simplification of state's space ensures feasibility of the problem in the used Q-learning algorithm [9]. For each state $s_i \in S$, being S the space of available CPU usage configurations, we introduce $A(s_i)$ as the set of valid actions. The agent can choose if increasing maximum CPU usage threshold in HPA autoscaler, decreasing it or maintaining it, so the set of all action is defined as $A = \{-1, 0, +1\}$. If the agent reaches the minimum or maximum configuration allowed, the action set $A(s_i)$ is reduced by removing the invalid action for that state, *e.g.*, $a = +1$ for $s_i = 90$. To evaluate the chosen action at each iteration, the agent receives a reward based on the performance achieved through the action at the specific iteration. This reward is usually based on the distance between the obtained performance and a specific *Service Level Agreement*, such as a response time target value [9]. The target value can be identified using adaptive, dynamic approaches such as metalearning. Nevertheless, for this preliminary study we considered the minimal latency value obtained in the baseline case, *i.e.*, 0.0341653 s, as the target *Service Level Agreement*, including a tolerance for greater values. Therefore, the calculation of the immediate reward r_i for iteration i is defined as follows:

$$r_i = \begin{cases} \frac{target}{result_i} * 10 & \text{if } result_i \leq target * 1.05 \\ 1 & \text{otherwise} \end{cases} \quad (2)$$

It has to be noted that, being $result_i$ the obtained latency for iteration i and $target$ the *Service Level Agreement* value, the lower is $result_i$ with respect to $target$, the higher the reward. The Q-learning process is configured with the following parameters: learning rate $\alpha = 0.5$ to balance new and past information, a discount factor $\gamma = 0.95$ to ensure that the agent promotes a long-term high return. Finally, an ϵ-greedy policy is implemented for which the agent starts in full exploration mode and opts for exploitation of its previous experience with increasing probability. Therefore, an ϵ is set to 1 for the initial 10 iterations and will be gradually decreased to 0.2, fixing that value from iteration 150 to 200. During the last 50 iterations the agent only relies on the learned Q-table to choose its actions ($\epsilon = 0$).

The results of the training process are depicted in Fig.6. It can be seen how the RL agent gradually explores the available configurations, tending to oscillate around the interval between 30% and 50% CPU usage and 80% CPU usage setting, being the local minimum previously shown in Fig.5. Moreover, the lower the ϵ coefficient gets, the stronger gets the trend to choose in the 30%-50% interval, up to the final convergence to the optimal value of 30%, when the agent is relying exclusively on the acquired knowledge to choose the next action. The evolution of the obtained average latency during the load test at each iteration is reported as a blue line in Fig.6. The latency trend during the training process is kept in the interval between 0.035 s and 0.055 s, with few peaks over the maximum. From around the 175^{th} iteration onwards, there is an increased tendency to get values of 0.04 s and below, gradually decreasing the resulting latency when the agent is set for a 100% exploitation mode (no random choices, $\epsilon = 0$).

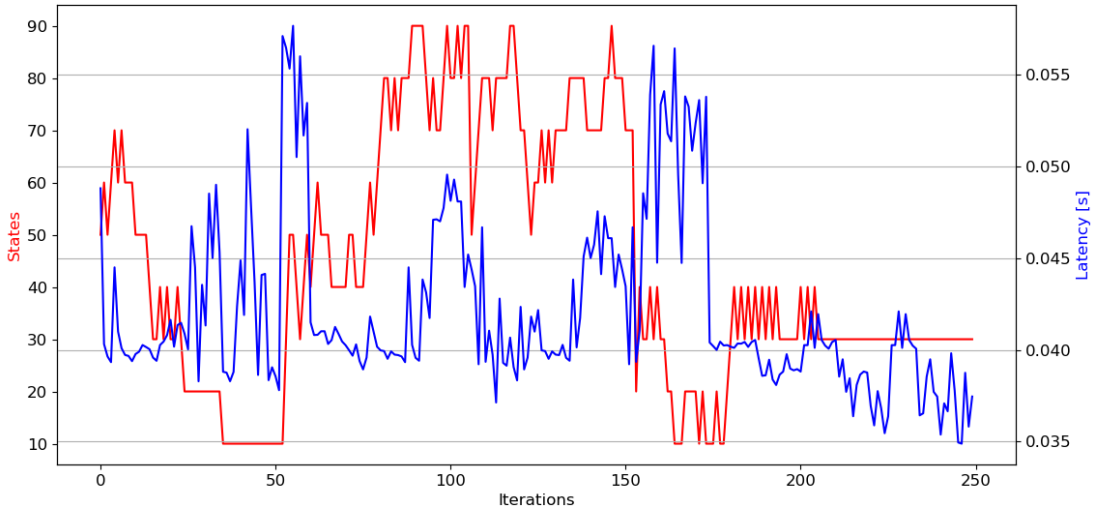


Fig. 6: Q-learning model state transition during training process (red), average latency evolution during training (blue).

Indeed, in this phase, apart from oscillations of hundredths of a second due to system's hidden stochastic conditions, a trend toward lower latencies can be observed. In the last 10 iteration of the training process, latencies are oscillating around 0.0375 s, reaching a minimum of 0.03488667 s, 0.7 ms greater than the minimum observed in the baseline case (Section V).

VII. Conclusion

The optimization of dynamic resource provisioning (auto-scaling) in serverless applications has become a key area of research that led to the introduction of various auto-scaling mechanism, mainly partitioned in workload-based and resource-based auto-scaling solutions. In this work, we investigated, with the use of a preliminary simplified model, the applicability of reinforcement-learning algorithms to learn effective resource-based auto-scaling policies for OpenFaaS, the highly adopted open-source serverless platform. The analysis was conducted considering a testbed with limited hardware resources to simulate an edge-computing scenario, a frequent use-case for serverless technologies. The experiment showed the ability of an RL-model to learn the optimal CPU usage threshold configuration for resource-based auto-scaling in a limited number of iterations. Nevertheless, these results are presented as preliminary because of resource-based auto-scaling features variability: Resource-based auto-scaling mechanisms are influenced by the impact of the specific serverless platform's components resource usage. Moreover, as noted in the baseline experiment, when the CPU usage threshold is close to the maximum in our edge-like testbed, the cluster incurs into strong instabilities. In such situation where the system is overloaded, the resource-usage footprint of the cluster's monitoring components can influence negatively the service quality. Hence, we plan to further investigate these factors to develop an enhanced RL model for CPU usage based auto-scaling optimization, taking in account these features influencing system conditions in the modelled states. Moreover, we plan to test the trained model in a dynamic scenario with time-varying workloads made by CPU-intensive functions, to further refine the agent in a real-time changing environment scenario, without relying on a baseline case to evaluate its performance. Nevertheless, we highlight the flexibility of this type of research: being resource-based auto-scaling in serverless platform based on the underlying Kubernetes HPA functionalities, the procedures shown in this work for OpenFaaS can be applied in other serverless Kubernetes-based platforms.

Acknowledgment

This work has been partially supported by the EU project 5G-CARMEN under GA No. 825012. The views expressed are those of the authors and do not necessarily represent the project. The Commission is not responsible for any use that may be made of the information it contains.

References

[1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, p. 44–54, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3368454

[2] "CNCF WG-Serverless Whitepaper v1.0."

[3] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan, "A serverless real-time data analytics platform for edge computing," *IEEE Internet Computing*, vol. 21, no. 4, pp. 64–71, 2017.

[4] A. Glikson, S. Nastic, and S. Dustdar, "Deviceless edge computing: Extending serverless computing to the edge of the network," in *Proceedings of the 10th ACM International Systems and Storage Conference*, ser. SYSTOR '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3078468.3078497

[5] J. Li, S. G. Kulkarni, K. K. Ramakrishnan, and D. Li, "Understanding open source serverless platforms: Design considerations and performance," in *Proceedings of the 5th International Workshop on Serverless Computing*, ser. WOSC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 37–42. [Online]. Available: https://doi.org/10.1145/3366623.3368139

[6] "Kubernetes horizontal pod autoscaler." [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[7] S. K. Mohanty, G. Premsankar, and M. di Francesco, "An Evaluation of Open Source Serverless Computing Frameworks," in *CloudCom 2018*, 2018, pp. 115–120.

[8] "OpenFaaS: Open function as a service," [Accessed on 26-Sep-2021]. [Online]. Available: https://www.openfaas.com/

[9] L. Schuler, S. Jamil, and N. Kuhl, "Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 804–811. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CCGrid51090.2021.00098

[10] "Prometheus - monitoring system & time series database." [Online]. Available: https://prometheus.io/

[11] B. Jang, M. Kim, G. Harerimana, and J. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. PP, pp. 1–1, 09 2019.

[12] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach Learn*, vol. 8, p. 279–292, May 1992.

[13] L. Baresi and D. Filgueira Mendonça, "Towards a serverless platform for edge computing," in *2019 IEEE International Conference on Fog Computing (ICFC)*, 2019, pp. 1–10.

[14] P. Benedetti, M. Femminella, G. Reali, and K. Steenhaut, "Experimental analysis of the application of serverless computing to IoT platforms," *Sensors*, vol. 21, no. 3, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/3/928

[15] V. Aggarwal and B. Thangaraju, "Performance Analysis of Virtualisation Technologies in NFV and Edge Deployments," in *IEEE CONECCT 2020*, 2020, pp. 1–5.

[16] A. Palade, A. Kazmi, and S. Clarke, "An evaluation of open source serverless computing frameworks support at the edge," in *2019 IEEE World Congress on Services (SERVICES)*, vol. 2642-939X, 2019, pp. 206–211.

[17] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck, "From data center resource allocation to control theory and back," *Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010*, pp. 410–417, 07 2010.

[18] C. Bitsakos, I. Konstantinou, and N. Koziris, in *DERP: A Deep Reinforcement Learning Cloud System for Elastic Resource Provisioning*, 12 2018, pp. 21–29.

[19] S. Horovitz and Y. Arian, "Efficient cloud auto-scaling with sla objective using q-learning," in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2018, pp. 85–92.

[20] J. Rao, X. Bu, C.-Z. Xu, and K. Wang, "A distributed self-learning approach for elastic provisioning of virtualized cloud resources," 07 2011, pp. 45–54.