MLOps in the Metaverse: Human-Centric Continuous Integration

Ningxin Su, *Graduate Student Member, IEEE*, and Baochun Li, *Fellow, IEEE*

***Abstract*— The metaverse is a virtual world that exists entirely in a computer-generated environment, and it offers a new frontier for machine learning. One of the major challenges for using machine learning in the metaverse is MLOps (Machine Learning Operations), an emerging field that focuses on deploying and managing machine learning models in production. It has been widely acknowledged that machine learning models require a large amount of data to learn and make accurate predictions, and such data is generated progressively in real-time as human users interact with the metaverse. Due to the human-centric nature of the metaverse, it goes without saying that, once deployed, models need to be able to adapt to the constantly changing interactive environment and still make accurate predictions. Borrowing a page from software engineering, in this paper, we explore the design space of human-centric *continuous integration* in metaverse environments, where labeled data samples accumulated with explicit human interactive behavior (e.g., using virtual reality or augmented reality headsets) are used for fine-tuning a deployed deep learning model over a sustained period of time. We propose SPIN, a new mechanism that efficiently utilizes data samples collected from a large number of participating human users over time to fine-tune a deployed model that is shared across all the users. In an extensive array of experimental results using image classification and state-of-the-art `YOLOv8` object detection models as case studies, we show that SPIN outperforms FedBuff, a state-of-the-art asynchronous FL mechanism from conventional federated learning, by a substantial margin.**

***Index Terms*— Continuous integration, MLOps, federated learning, metaverse.**

I. INTRODUCTION

AS AN emerging technological trend, the metaverse is an interactive digital environment that is expected to revolutionize the way people interact by enabling real-time, immersive communication via digital avatars [1]. Needless to say, the implementation of engaging features in the metaverse requires the assistance of machine learning models on a wide variety of tasks, such as object detection, in real-time. As such, machine learning will become a fundamental building block in many metaverse applications. Most of these applications will use pretrained models in their deployment in the metaverse, and such deployed models need to be monitored and fine-tuned to ensure that they are performing as expected, taking into account the constantly changing environment in the metaverse due to human interactions. Generally referred to as *MLOps* (Machine Learning Operations), applications need to have a process in place to manage the lifecycles of its models, including version control due to model fine-tuning, testing, and retraining.

As challenging as MLOps are in general-purpose production environments, we argue that the challenges MLOps pose in the metaverse are unique and require special attention. In the metaverse, due to its unique interactive nature involving human participants, human decisions and behavior produce data interactively and progressively over a sustained period of time, sometimes days or even weeks. Such data, which is generated through sporadic human interactions in the metaverse, can be collected and utilized for various purposes. For instance, users can label objects in a sequence of images from a live video feed using their headsets and over-the-air gestures from time to time. This data can then be employed to train an object detection model or easily fine-tune an image classification model in a specific area. For example, members in a family can harness devices in the metaverse to accumulate their own data, which can subsequently be used to train a personalized butler system exclusively catering to the needs of the family members within their house. As new data that reflect and arise from human intervention in the metaverse are therefore produced in a sporadic fashion, they could — and should — be utilized to fine-tune and improve the quality of deployed models, as an important feature of MLOps in the metaverse.

Yet, due to the human-centric and sporadic nature of data generation over the prolonged period of time, it is non-trivial to utilize such data for fine-tuning deployed models in the metaverse. For example, given that the metaverse involves sensitive personal information, it is crucial to guarantee the privacy of user-generated data in a decentralized metaverse environment [2]. In the literature, *federated learning* (FL) [3] has been long used for preserving such data privacy generated by human users on the edge devices. Conventionally, federated learning is a distributed learning paradigm where multiple edge devices, such as smartphones or IoT devices, collaboratively train a shared machine learning model without sharing their raw data with each other or with the central server. Though it is not perfect in preserving data privacy and several challenges still persist, federated learning has been widely acknowledged to reduce the risk of exposing sensitive user data.

Manuscript received 15 March 2023; revised 1 August 2023; accepted 31 August 2023. Date of publication 1 January 2024; date of current version 1 March 2024. *(Corresponding author: Baochun Li.)*

The authors are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 1A1, Canada (e-mail: ningxin.su@mail.utoronto.ca; bli@ece.toronto.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2023.3345385.

Digital Object Identifier 10.1109/JSAC.2023.3345385

There are, however, a number of challenges that we need to address when utilizing the conventional FL paradigm for fine-tuning deployed models in a human-centric metaverse. As human users interact with the metaverse, labeled data samples used for such fine-tuning are produced sporadically and unpredictably over time. Such a *human-centric* paradigm is remarkably different from typical assumptions used in conventional FL, where client data is assumed to be readily available *in each communication round* that the client is selected by the server. In the conventional FL literature, data is assumed to be heterogeneous and not independent and identically distributed (non-i.i.d.), which remains a valid assumption in the metaverse. However, the literature has not considered the *timing* when data samples are produced, and has not studied the cases where private data from human users arrives over time sporadically. Additionally, local data samples in human-centric metaverse environments may no longer be produced in the hundreds of thousands and in batches, but rather are only generated sporadically in small increments. In such cases, conventional FL may not operate effectively, simply because clients may not have the data needed for fine-tuning a deployed model when they are selected by the server, and conversely, when data is produced on a client due to human interactions, the server may not be selecting it in a timely fashion. This can significantly impact the performance of fine-tuning a deployed model.

In this paper, we propose SPIN, a new mechanism designed specifically to fine-tune globally deployed models by taking full advantage of data samples generated sporadically in a human-centric metaverse. Borrowing a page from modern software engineering, SPIN adopts the design principles of *continuous integration* (CI), which is a software engineering practice that involves integrating code changes frequently and automating the build and deployment processes. Similar to how continuous integration promotes collaboration across team members frequently rather than waiting till the end of a development cycle, SPIN recognizes the fact that data is generated unpredictably and sporadically over time, and is designed to incorporate recently produced data in a timely fashion rather than waiting till a future communication round, so that human users can collaborate in the model fine-tuning process.

SPIN is designed for MLOps in the metaverse, in the sense that it targets fine-tuning deployed models that have been pretrained, rather than training models from scratch as in conventional FL. Similar to conventional FL, the server aggregates the *push updates* from clients after their local training is completed and a new version of the deployed model is produced. Rather than waiting for server selection in conventional FL, however, clients — who are human users in the metaverse — proactively request the deployed model from the server whenever a sufficient amount of new data becomes available. We call these explicit requests *pull requests*, as the globally deployed model will be sent only when a client requests it. Just like in software engineering, SPIN deals with potential conflicts between pull requests from some clients and push updates from other clients. When resolving such clients, we must decide whether we should send the current model immediately to reduce the response time, or wait for the push updates to be received so that a newer version of the model can be used for training. For privacy protection, just like in conventional FL, all user data remains local to the device, and only model parameters are transmitted between the server and users.

Our original contributions to this paper are three-fold. *First*, we are the first to consider MLOps and continuous integration in the metaverse, and bring "human-in-the-loop" into the fine-tuning process of pretrained models that have been deployed. *Second*, we propose SPIN, a new mechanism that integrates the data privacy benefits of conventional FL and the timeliness of continuous integration for sporadically generated data in the metaverse with human interactions over time. In particular, SPIN resolves potential conflicts between *pull requests* from some clients and *push updates* from others, with the design objective of improving the convergence performance by considering the staleness of models sent to clients in response to their pull requests. With a mathematically rigorous analysis of its convergence behavior, we prove that convergence is guaranteed with our proposed mechanism. *Finally*, we have implemented SPIN in the PLATO FL research framework, and present a comprehensive evaluation of its performance. Our evaluations start with image classification tasks with `LeNet-5` and `ResNet-18` models as benchmarks, using `CIFAR-10` and `EMNIST` datasets. The upshot of our experimental evaluation is its use of the state-of-the-art `YOLOv8` object detection model to best reflect real-world human-centric tasks, in which human users label new objects in their headset's video feeds, using paired virtual keyboards and over-the-air gestures. Our experimental results show that SPIN converges efficiently when fine-tuning pretrained models with sporadic and small-sized data generated by human behavior, and outperformed FedBuff [4], a state-of-the-art asynchronous FL mechanism, across all datasets and models. For the best possible reproducibility, our experimental settings and source code will be made open-source as published examples in the git repository for the PLATO FL research framework.

II. PRELIMINARIES

A. Machine Learning Operations in the Metaverse

The metaverse, a virtual world where users can interact with each other and digital objects, is rapidly growing in popularity and complexity. As the metaverse becomes more sophisticated, the need for intelligent systems that can adapt and learn from user behavior increases. Machine learning is a powerful tool that can be used to enhance various aspects of metaverse applications.

One of the primary ways that machine learning can be used in metaverse applications is to improve the user experience. By analyzing user behavior, machine learning algorithms can learn patterns and preferences, allowing metaverse applications to tailor their experiences to individual users. For example, machine learning algorithms can analyze user interactions with virtual objects, such as avatar movements, and use this data to improve animations or suggest new virtual objects to users. As another example, machine learning can be used to optimize the performance of metaverse applications, allowing

for smoother gameplay and more immersive experiences. Such applications of machine learning could aid in creating more engaging and realistic experiences for users in the metaverse [5], [6]. Overall, machine learning has a crucial role to play to allow for more personalized and immersive user experiences in the metaverse, and to create new opportunities for users and developers alike [7], [8], [9], and [10].

Existing research related to enabling users to participate in training machine learning models in the metaverse is primarily focused on three challenges: *resource allocation*, *privacy protection*, and *incentive mechanisms*. With respect to *allocating resources*, Cong et al. [11] proposed a resource allocation framework that minimized the cost of a virtual service provider, using a two-stage stochastic integer programming technique. The framework incorporated uncertainty when modeling user demands, allowing the virtual service provider to make informed decisions under unpredictable conditions. Yue et al. [12], on the other hand, proposed a dynamic resource allocation approach using evolutionary game theory. In addition, Nguyen et al. [13] was proposed as a blockchain-based framework that efficiently managed resources and encouraged user contribution for the metaverse, utilizing smart contract mechanisms and a sharding scheme to enable automated interactions between the metaverse service provider and its users.

Unfortunately, none of the existing works have investigated the core challenge involved with MLOps in the metaverse, *i.e.*, when machine learning models are deployed and need to be fine-tuned over time. Such fine-tuning is necessary to adapt to a human-centric metaverse environment, and relies upon data samples progressively generated by the human user over a prolonged period of time. In this paper, we propose to adopt the design principles of *continuous integration* from software engineering, and design a new mechanism, SPIN, to facilitate such a fine-tuning process.

B. Federated Learning

There exists a vast amount of literature on conventional federated learning (FL) [3], a distributed machine learning paradigm proposed to address privacy concerns by training on client devices only and aggregating updates on the server. Starting from its inception, conventional FL is designed with one core mechanism as its foundation: in each communication round, the server selects its clients from all available devices using a client selection algorithm, and sends its current global model to the selected clients only. Such a core mechanism in conventional FL naturally assumes that if a client is selected, its local data samples are readily available to commence local training immediately. Unfortunately, this is not suitable for our design objective of continuous integration when fine-tuning deployed models for MLOps in the metaverse, as data samples on each client are produced due to human behavior, and such events are both *sporadic* and *unpredictable* in nature.

One may argue that the *asynchronous* design of conventional FL, first proposed in FedAsync [14] and then tested and validated with large-scale experiments in FedBuff [4], may be a more suitable design for our design objective of continuous integration. Asynchronous FL proposes to start aggregating client updates on the server after a minimum number of client updates arrives, yet *before* receiving all the updates from selected clients. This is typically advantageous in situations where clients have heterogeneous training capabilities, and some slower clients may finish local training at a much later time. However, as the server still selects new clients in the next round of communication in asynchronous FL, it would still not be ideal for the human-centric fine-tuning task we are considering in the metaverse, since the selected clients may not have data readily available at all times.

Yang et al. [15] proposed AFL, which allowed clients to participate in the FL training process by pulling the global model proactively from the server with the local timestamp. This is a much more suitable design for our human-centric fine-tuning task, but it misses several design elements in the MLOps fine-tuning process we study in this paper. *First*, it does not consider sporadic arrivals of events when client data becomes readily available, and how such arrival processes affect convergence behavior. *Second*, just like the conventional FL literature, it does not specifically consider the task of fine-tuning deployed models that have been pretrained. *Finally*, it does not consider the potential conflicts between pull requests from some clients and push updates from others to the server, or how such conflicts should be best resolved. In contrast, SPIN is designed specifically for our human-centric fine-tuning task in the metaverse, and used a state-of-the-art object detection model, `YOLOv8`, to validate its effectiveness in the metaverse.

Last but not least, the task of fine-tuning pretrained models has been empirically and theoretically investigated in the recent FL literature. Nguyen et al. [16] empirically studied the impact of starting from a pretrained model in federated learning using several benchmark datasets, and showed that not only did it reduce the training time, but it also led to more accurate models as compared to starting from random initialization. It was also found that starting from a pretrained model reduced the effect of both data and system heterogeneity. From a more theoretical perspective, Tan et al. [17] proposed a mechanism for clients to jointly learn to fuse the representations generated by multiple fixed pre-trained models, rather than training a large-scale model from scratch, so that clients with lower computational resources can still participate. In this paper, we recognize the fact that clients may have much lower computational resources than typical servers, which are only sufficient for fine-tuning a deployed global model using a small amount of data samples. In addition, we focus on the more practical situation that data samples are generated by human users, and can only arrive sporadically over a long period of time. SPIN is designed to take advantage of such sporadically generated data for fine-tuning a pretrained model in the metaverse, using client devices that have low computational resources.

III. SPIN: Problem Formulation and Algorithm Design

In this section, we begin by formulating the problem of continuous integration and fine-tuning deployed models in the

context of MLOps in the metaverse. We then present the technical details of SPIN, our new algorithm designed to solve our formulated problem in the metaverse context.

A. Problem Formulation

The metaverse is a virtual world that exists entirely within computer simulations, and as such, human users interact with it using their virtual reality headsets, augmented reality glasses, and other suitable input devices such as virtual keyboards and over-the-air gestures. Consider an MLOps scenario that a state-of-the-art object detection model, such as the Ultralytics `YOLOv8`, is already pretrained and readily deployed. It is, however, pretrained using data that is not necessarily inclusive of the type and shape of objects that a human user interacts with in its metaverse, and as a result the deployed model may not be accurate detecting and classifying objects that a human user is interested in.

In this scenario, developers can design the metaverse application accordingly and ask human users to manually identify and label these objects that the object detection model is not trained on. As such manually labeled data samples accumulate at each user, our objective in this paper is a matter of MLOps in the metaverse context: how do we properly utilize these data samples to fine-tune the globally deployed model that is shared across all users in the metaverse, yet without compromising data privacy?

Conceptually, though its efficiency may be questionable, conventional federated learning (FL) can be used to achieve such an objective. In order to design a new algorithm that best achieves such an objective in general, we define several research objectives that are more specific:

- ♢ *Convergence.* Just like in conventional FL, the fine-tuning process must converge to a steady state with respect to the validation accuracy of the globally deployed model.
- ♢ *Efficiency.* One of the key challenges in fine-tuning a globally deployed model is how decentralized data samples produced by different users can be utilized efficiently in the training process.
- ♢ *Timeliness.* As data is only produced sporadically over a long period of time, it is important to ensure that once data samples have been generated through human interaction in the metaverse, they are utilized as quickly as possible.

With these objectives in mind, we are now ready to present the technical details of SPIN, our new mechanism designed for continuous integration in the MLOps fine-tuning process.

B. SPIN: Algorithm Design

1) Continuous Integration: Continuous integration (CI) is a software engineering practice that involves integrating code changes in a fully automated way into a shared repository, so that all changes can be incrementally tested without waiting till the end of a development cycle. Borrowing the general design philosophy of continuous integration, and given the objective of *timeliness* for integrating the data samples produced by human intervention, we also wish to utilize data samples as soon as possible after they are produced over time.

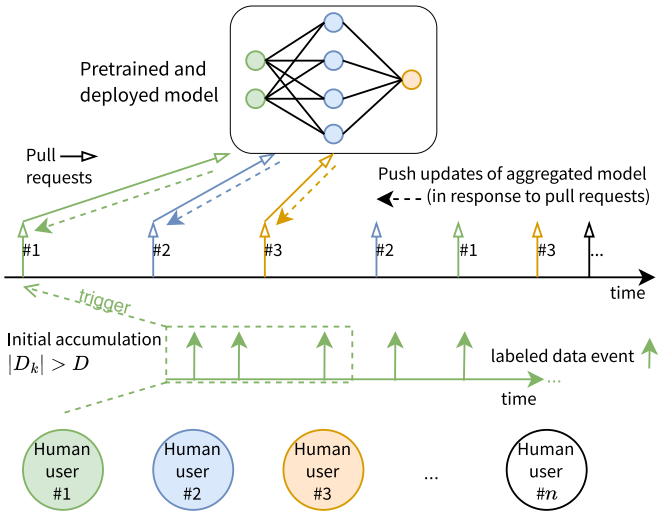


Fig. 1. An overview of data generation events, pull requests and push updates in SPIN.

As its foundation, SPIN inherits the design principles of both continuous integration and conventional FL. We assume a large number of human users in the metaverse, all of whom interact with the metaverse and produce labeled data samples in a sporadic fashion. As the number of potential clients is large, conventional FL opts to select a *subset* of clients in each communication round and to send the global model to the selected clients only. In SPIN, rather than actively selecting clients in each round, the server responds to *pull requests*, as we elaborate next.

2) Client Pull Requests: Similar to workers in AFL [15], in SPIN, clients *proactively pull* the globally deployed model from the server, and the server sends its current model only to the clients who sent these pull requests. This helps mitigate the negative effects on timeliness when the server selects clients in each communication round in conventional synchronous or asynchronous FL.

Different from workers in AFL, however, clients are not allowed to pull the global model arbitrarily at any time, and would not be able to completely take control of their own optimization process. As data samples are produced only sporadically as human users interact with the metaverse, the quantity of such data samples produced at each client would be small, at least initially. When local datasets are small, locally trained models can vary greatly from a globally deployed model. Local models with poor quality, however, can arbitrarily deteriorate the aggregate model quality, causing convergence processes to fail in these settings. To mitigate the problem of small local datasets, it has been proposed in recent literature [18] that local models can be redistributed across clients through the server in a permutation step, in order to allow each local model to train on a daisy chain of local datasets. In practice, however, such a complex permutation mechanism can lead to a significant amount of communication overhead, which is inconsistent with our objective of timeliness when utilizing freshly produced data samples.

When should a client send its pull requests to the server, then? As illustrated in Fig. 1, in SPIN, before sending its

initial pull request, a client is asked to accumulate a minimum number of data samples that exceed a pre-determined threshold that is task-dependent. For example, for object detection tasks using a globally deployed `YOLOv8` model, a client is asked to accumulate 64 data samples before sending its initial pull request. After the initial pull request is sent, a client sends a *new pull request* whenever a new data sample is produced by a human user, and this new data sample will participate in the ensuing local training process, along with all existing data samples that have been produced so far. This mechanism in SPIN is designed to mitigate the risk of convergence failures due to very small datasets, while maximizing timeliness in the event that a new data sample arrives. A complete description of SPIN's client algorithm is shown in Algorithm 1.

Algorithm 1 SPIN: Client Algorithm at User k

Require: The set of locally produced data samples D_k; a minimum threshold D; the batch size B, the learning rate η, and the number of training epochs E.

while `True` **do**
  Data samples are sporadically produced and added to D_u, accumulated with human interaction in the metaverse
  if $|D_k| \geq D$ **then**
    Send a new *pull request* to server S and wait for the deployed model $\boldsymbol{w}_s$
    if model $\boldsymbol{w}^s$ has been received **then**
      $\mathcal{B} \leftarrow$ (split D_k into batches of size B)
      for each local epoch $j \leftarrow 1$ to E **do**
        for batch $b \in \mathcal{B}$ do **do**
          $\boldsymbol{w}_k \leftarrow \boldsymbol{w}_k - \eta \nabla \ell(\boldsymbol{w}_k, b)$
        end for
      end for
      Submit a *model update* $\boldsymbol{w}_k$ to the server
    end if
  end if
end while

3) Server Algorithm: We are now ready to explore the design space for the SPIN server algorithm, which governs how the server handles the pull requests it receives from clients who have new data samples produced locally.

As Fig. 1 shows, after a pull request has been received by the server, it will send a push update in response, containing the currently deployed model. Since client devices — such as headsets — in the metaverse are computationally much less powerful than servers (without ready GPU access, for example), it should naturally be assumed that their local model fine-tuning processes using local data will take a considerable amount of time. As a real-world example, with an M1 CPU on a modern Mac computer, approximately 5-6 minutes will be needed to fine-tune a pretrained `YOLOv8` object detection model with 128 data samples and 5 epochs.

But why would it matter for slow client devices in the metaverse to finish their local fine-tuning processes over a longer period of time? Let us consider a situation where a client has just finished its local fine-tuning process, and sent its updated model back to the server, as in conventional FL. Should the server aggregate the update into its current model? Based on state-of-the-art work in the literature on asynchronous FL [4], [19], convergence may fail if the server chooses to aggregate a model update from a client as soon as it is received (which was advocated by FedAsync [14]). Instead, previous work, such as FedBuff [4], advocated for the server to wait for a minimum number of model updates before aggregating them into its current model, using either a conventional aggregation algorithm such as Federated Averaging (FedAvg) [4], or a staleness-aware aggregation algorithm, such as PORT [19].

However, a fundamental difference between SPIN and conventional FL is that in SPIN, the server may be receiving more *pull requests* while it is waiting for the minimum number of model updates to arrive. This is particularly the case if it takes a rather long time — such as several minutes — for a client to finish its local fine-tuning process. In conventional FL, before the current round of server aggregation completes, no clients will be selected since the next round has not yet started. In SPIN, new pull requests may arrive at the server at any time. It is straightforward to see that the server's strategy of waiting for a minimum number of model updates makes the problem of *model staleness* worse in SPIN: a client is said to be using a *stale* model if it started with a model that has an earlier version than the model into which its update is aggregated. In other words, more concurrency and asynchrony between clients lead to higher risks of model staleness.

Just as SPIN's algorithm design is inspired by continuous integration in software engineering, it is worth pointing out the analogy that the potential issue caused by model staleness resembles potential merge conflicts in git repositories. With a git repository, multiple clients are allowed to check out the current version; yet, if they concurrently work on the current version for a long period of time, conflicts may arise when merging their updates on the server, leading to the need for manual intervention. In the context of SPIN, such potential merge conflicts correspond to poor performance or failures in training convergence.

In the design space for the SPIN server algorithm, we can mitigate the negative effects of model staleness with two potential measures. *First*, when a pull request is received, the server may choose to delay sending the push update with the current model until some or all *outstanding model updates* — corresponding to clients who received push updates but are still performing their local fine-tuning processes — have been received. *Second*, the server may aggregate the model updates it has received sooner and in a more timely fashion, so that the time interval between two aggregation rounds is shorter and fewer clients would receive stale models.

We believe that the first alternative will not be effective in mitigating the model staleness problem. This is because that with pull requests arriving at any time, model staleness is mostly caused by the fact that a large number of pull requests are received by the server while many clients are still performing their fine-tuning. The more frequent pull requests that arrive and the longer it takes for clients to complete their fine-tuning, the more severe model staleness may be. Delaying a push update in response to a pull request does not reduce the time a client takes to perform fine-tuning or the frequency of pull updates; it simply gathers pull updates into batches, and

the overall training process becomes more synchronous with a sequence of communication rounds.

C. A Toy Example

In SPIN's design, we aim to maintain the asynchrony and concurrency of pull updates. Pull updates are sent to the server whenever human users occasionally generate labeled data samples. Upon receipt of a user's pull update, the server checks two parameters: the number of user updates received and the timeout π. As shown in Fig. 2, the server begins the aggregation process when it receives a minimum number of model updates within the timeout π period. In this case, the condition that triggers the server to start the aggregation process is the receipt of three model updates from the users, not a timeout. We do this to avoid model staleness or skew in the model after convergence due to too few updates being aggregated.

However, setting a timeout π is crucial. It's important to note that there's a significant variation in the speed at which users train their models, which is the reason for the heterogeneity in federated learning. Imagine a scenario where a certain region is at nighttime for a while, and user activity is not frequent. Some users who accumulate data and start the FL training process at night may have very limited computational resources, leading to considerable differences in training speeds. If the server waits for a long time for model updates and does not use the currently received model updates for aggregation and global model update, it would lead to a situation where fast-training users have already trained and sent model updates several times, but the server's pushed model remains unchanged. These users would receive models that are much stale than their previously completed models for training their new data. See Fig. 2 as an example, we assume only user 1 and user 2 have newly generated data and send pull requests to the server. However, user 1 trains several times faster than user 2. In practice, the minimum number of model updates received by the server can be set to 10 or more. By the time user 1 has completed several pull-push communications with the server, user 2 is still training. At this point, the server has not received the minimum number of updates from the users, but user 1 has already been training with a model several rounds older than the one they trained. To prevent fast-training users from wasting computational resources, we apply the timeout hyperparameter π to ensure that the model received by the client when a push update is sent in response to its pull request is no more than π versions older than the latest version.

The full description of the SPIN server algorithm is shown in Algorithm 2. The SPIN server algorithm does not pull updates in batches but waits for a minimum number of model updates before aggregating them, with a crucial hyperparameter added: timeout π. If the number of model updates has not reached the minimum threshold K before timeout π expires, the SPIN server will continue to aggregate all model updates received so far. We chose not to modify the client selection or aggregation algorithm in the FL process, as our metaverse users are very different from the traditional clients in federated learning. Their data distribution, data size, and training speed are significantly different, and as they are the first users of the metaverse, their data is valuable. For highly heterogeneous users, modifying the client selection or aggregation algorithm in the federated learning process can significantly improve accuracy. However, we do not want our performance improvement to be due to selecting certain users beneficial to the global model or giving more weight to certain users' models. We hope that in the metaverse, users and their data contribute fairly to the global model, at least during the initial construction of the metaverse world.

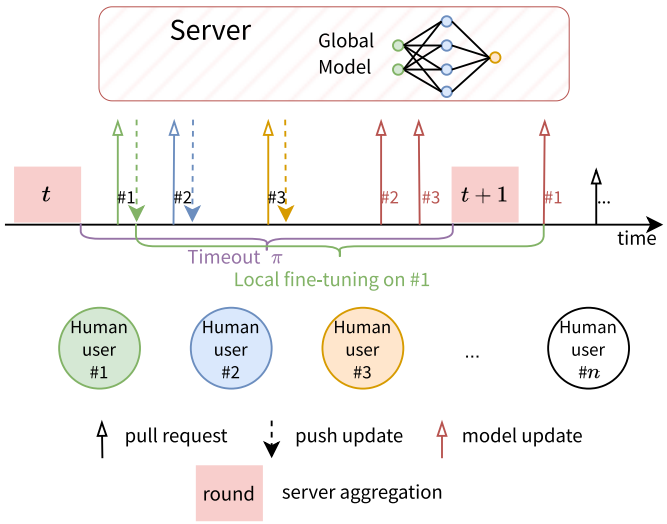


Fig. 2. The server algorithm in SPIN: server aggregation starts when a minimum number of model updates have been received, or when a timeout π expires. Whenever a pull request is received from a client, a push update with the current global model will be sent immediately.

Given our desire for each user to participate fairly in the construction of the metaverse world with their data, we hope to avoid conflicts or model regressions caused by concurrency and asynchrony in communication with the server when designing the algorithm. Just like FedAvg [3] and FedBuff [4], it is intentionally designed to be simple, as our experiments showed that additional complexity does not necessarily lead to corresponding performance improvements. We choose not to include more complexity in our design due to empirical observations in our experiments.

IV. CONVERGENCE ANALYSIS

In this section, we offer a thorough convergence analysis of the SPIN algorithm. A key challenge in SPIN is the inability to actively select users by the server, in contrast to traditional federated learning. Consequently, the time steps at which users engage in the federated learning process are not uniform, rendering our problem more closely related to asynchronous federated learning. In order to present a comprehensive analysis of the SPIN algorithm, it is essential to examine the convergence properties of the algorithm. Unlike traditional federated learning structures, SPIN does not utilize a fixed set of users. Instead, in our subsequent experiments in Section V, we will employ a Poisson process to simulate the event arrival times for pull requests to the metaverse server S. Given this

Algorithm 2 SPIN: Server Algorithm

Require: The globally deployed model $\boldsymbol{w}_s$; the minimum number of model updates K before aggregation starts; the timeout hyperparameter π.

while `True` **do**
  Wait for *pull requests* and model updates from clients
  if received a *pull request* from client k **then**
    Send the current model $\boldsymbol{w}_s$ in a *push update* to k
  end if
  if received a *model update* $\boldsymbol{w}_k$ from client k **then**
    Add $\boldsymbol{w}_k$ to the queue of *model updates* $\mathcal{Q}_s$
  end if
  if $|\mathcal{Q}_s| \geq K$ or timeout π exceeded **then**
    $\boldsymbol{w}_s \leftarrow \sum_k p_k \boldsymbol{w}_k$, where $p_k = \frac{|D_k|}{\sum_k |D_k|}$ ▷ FedAvg
  end if
end while

context, we aim to provide a convergence guarantee for SPIN in a smooth, non-convex setting.

Distinct from other federated learning paradigms, the pull requests and corresponding push updates within the SPIN framework are uncontrolled and asynchronous. This characteristic poses unique challenges in terms of convergence analysis. To address these challenges, the algorithm builds upon techniques from an existing asynchronous federated learning algorithm, PORT [19], in the literature.

In the asynchronous setting, the user updates are not synchronized, and the server must handle the received updates in an online fashion. To analyze the convergence of SPIN, we consider the non-convexity of the problem and the stochastic nature of user updates. We leverage the theory of stochastic approximation and the recent advances in non-convex optimization to derive convergence bounds for the algorithm.

In the efficiency mode, the server forgoes the traditional approach of waiting for a minimum of K updates from users for global aggregation. Instead, it performs the aggregation process within a predetermined time interval denoted as $\bar{t}$. As a consequence, the frequency and quantity of updates sent to the server by users become uncertain, resulting in a dynamically fluctuating count of updates eligible for aggregation. This scenario presents a new challenge of ensuring convergence when the user participation count K varies across communication rounds.

Without loss of generality, we assume that the number of updates transmitted to the server during $\bar{t}$ follows a Poisson distribution with a parameter r. This can be expressed as:

$$f(U; r) = \Pr(X = U) = \frac{\left(r\bar{t}\right)^k e^{-r\bar{t}}}{U!} \tag{1}$$

where U is the number of user updates within the unit time interval $\bar{t}$ and λ represents the average rate r of updates occurring per unit of time $\bar{t}$, denoting as $\lambda = r\bar{t}$.

We present the learning paradigm using specific notations. Each user u who receives the server's response performs E training epochs based on its local dataset D^u and the received global model $\boldsymbol{w}_i^u$, where i denotes the round index. For any local training epoch $j \in [0, E]$, the local model $\boldsymbol{w}_{i^u,j+1}^u$ is obtained by optimizing $\boldsymbol{w}_{i^u,j}^u$ using SGD with a batch size of B and a learning rate of η^j. This can be formulated as:

$$\boldsymbol{w}_{i^u,j+1}^u = \boldsymbol{w}_{i^u,j}^u - \eta^j g(\boldsymbol{w}_{i^u,j}^u), \tag{2}$$

where the gradient $g(\boldsymbol{w}_{i^u,j}^u) = \nabla f_u(\boldsymbol{w}_{i^u,j}^u, D^u)$. Once the time $\zeta = K\bar{t}$ is reached (which corresponds to π in SPIN), the server initiates the aggregation process for the received U^i updates. Due to the Poisson distribution property, the number of updates received by the server within ζ also follows a Poisson distribution with parameter $\lambda_\zeta = K\lambda$.

In this context, our convergence analysis is conducted under the following assumptions, which are commonly employed in prior work on federated learning analysis:

1) *L-smoothness*.
   Each objective function f_u of the user u is L-smooth. Thus its derivatives are *Lipschitz continuous* with constant L, i.e.,
   $$\| \nabla f_u(\boldsymbol{w}) - \nabla f_u(\boldsymbol{w}') \| \leq L \| \boldsymbol{w} - \boldsymbol{w}' \| \tag{3}$$
2) *Unbiased local gradient*.
   $$\mathbb{E}_\xi [f_u(\boldsymbol{w}, \xi)] = \nabla f_u(\boldsymbol{w}) \tag{4}$$
   where $\boldsymbol{w}$ denotes trainable parameters.
3) *Uniformly bounded local gradient*.
   The expected squared norm of stochastic gradients is uniformly bounded, i.e.,
   $$\mathbb{E} \| \nabla f_u(\boldsymbol{w}, \xi) \|^2 \leq G^2, u \in \{1 \dots U^i\} \tag{5}$$
4) *Bounded local gradient*.
   Let ξ be sampled from the u-th device's local data uniformly at random. The variance of stochastic gradients in each device is bounded as:
   $$\mathbb{E}_\xi \| f_u(\boldsymbol{w}, \xi) - f_u(\boldsymbol{w}) \|^2 \leq \sigma_u^2, u \in \{1 \dots U^i\} \tag{6}$$
5) *Bounded divergence between local and global gradient*.
   For any user k and the parameter $\boldsymbol{w}$, we define δ_u as an upper bound of $\| f_u(\boldsymbol{w}) - f(\boldsymbol{w}) \|^2$, i.e.,
   $$\| f_u(\boldsymbol{w}) - f(\boldsymbol{w}) \|^2 \leq \delta_u^2 \tag{7}$$

Lemma 1: Given U^i following the Poisson Distribution, the Bounded local gradient and Bounded divergence between local and global gradient over these clients can be defined as:

$$\sigma_l^2 := \sum_{u=1}^{\lambda_\zeta} p^u \sigma_u^2, \quad \delta_g^2 := \sum_{u=1}^{\lambda_\zeta} p^u \delta_u^2 \tag{8}$$

Proof: In each iteration i, there are U^i updates received by the server. Following assumption 4, the sum of local gradients $\sigma_{l^i}^2 = \sum_{u=1}^{U^i} \frac{|D^u|}{|D|} \sigma_u^2$. As U^i is not as random as previous works, such as [19], but is distributed according to a Poisson distribution $P(\lambda)$. This leads to the corresponding expectation

$$\sigma_l^2 = E_{i \sim P(\lambda)} E_\xi \left[\sigma_{l^i}^2\right] = E_{i \sim P(\lambda)} \sum_{u=1}^{U^i} p^u \sigma_u^2.$$

As $E_{U^i \sim P(\lambda_\zeta)} = \lambda_\zeta$, we can safely replace the U^i with its expectation term.

Similar derivations can be obtained for δ_g^2. □

Theorem 1 (Convergence Rate): Under the condition that the server performs global aggregation every time interval ζ, in which the number of arrival updates follows the Poisson Distribution $P(\lambda)$ defined by Eq. 1, the convergence rate for such dynamic users arrival will be:

$$\begin{aligned}\frac{1}{T}\sum_{i=0}^{T-1}\mathbb{E}\parallel\nabla f(\boldsymbol{w}_i)\parallel^2 \leq\ & 2\frac{(f(\boldsymbol{w}_0)-f(\boldsymbol{w}^*))}{\phi(E)TK}\\ &+6K\chi(\lambda_\zeta)L^2E\psi(E)\left(K^2\Omega^2+1\right)(\lambda_\zeta)^2\sigma^2\\ &+L\frac{\psi(E)}{K\phi(E)}(\lambda_\zeta)^2\sigma_l^2 \end{aligned} \tag{9}$$

where

$$\begin{aligned}\phi(E)&=\sum_{j=1}^{E}\eta^j, \psi(E)=\sum_{j=1}^{E}\left(\eta^j\right)^2\\ \chi(\lambda_\zeta)&=\sum_{u=1}^{\lambda_\zeta}p_u^2\\ \sigma^2&=\lambda_\zeta^2\sigma_l^2+\lambda_\zeta^2\delta_g^2+G^2.\end{aligned}$$

Additionally, to achieve the convergence upper bound, the relations between λ_ζ and η should follow:

$$\frac{4}{\chi(\lambda_\zeta)}\lambda_\zeta\eta_l^0\leq\frac{1}{L}. \tag{10}$$

Proof: Our proof follows the standard convergence procedure used in asynchronous federated learning methods, such as the one presented in [19], for non-convex objective functions. Before we start, we introduce S^u, which is the staleness bound for user u. S^u denotes the difference between the global model round and the user's update round. We start by using the L-smoothness assumption to establish an upper bound of $f(\boldsymbol{w}_{i+1})$, which is

$$\begin{aligned}f(\boldsymbol{w}_{i+1})\leq f(\boldsymbol{w}_i)&\underbrace{-\sum_{u\in U^i}p_i^u\langle\nabla f(\boldsymbol{w}_i),\triangle_{i^u}\rangle}_{T_1}\\ &\underbrace{+\frac{L}{2}\parallel\sum_{u\in U^i}p_i^u\triangle_{i^u}\parallel^2}_{T_2}\end{aligned} \tag{11}$$

where $\triangle_{i^u}=\sum_{j=1}^{E}\eta^j\nabla f_u\left(w_{i^u,j}^u\right)$, which implies that client u at round i sends the update $\triangle_{i^u}$ to the server.

The overall logic of the proof adheres to a classical framework for staleness-aware asynchronous federated learning. Nonetheless, this paper introduces a more comprehensive scenario wherein the count of updates transmitted to the server may be dynamic, hence introducing a variable denoted as U^i, which is dependent on the iteration index i.

Bound $\mathbb{E}[T_1]$.

$$\begin{aligned}E[T_1]&=-E\left[-\sum_{u\in U^i}p_i^u\langle\nabla f(\boldsymbol{w}_i),\triangle_{i^u}\rangle\right]\\ &=-E\left[\sum_{u=1}^{U^i}p_i^u\sum_j\eta_j\langle f(\boldsymbol{w}_i),f_u\left(w_{i^u,j}^u\right)\rangle\right]\\ &=-\sum_{u=1}^{\lambda_\zeta}p_i^uE\left[\sum_j\eta_j\langle f(\boldsymbol{w}_i),f_u\left(w_{i^u,j}^u\right)\rangle\right]\end{aligned} \tag{12}$$

The derivation of the third equation relies on the Poisson process as shown in Eq. 1.

Subsequently, conditioned on the Poisson process of arrival updates, the expectation in Eq. 12 can be expressed as:

$$\begin{aligned}\mathbb{E}[T_1]=&-\frac{1}{2}\lambda_\zeta\left(\sum_{q=0}^{Q-1}\eta^q\right)\|\nabla f(w_i)\|^2\\ &+\sum_{q=0}^{Q-1}\frac{\lambda_\zeta\eta^q}{2}\left(-\mathbb{E}_{\mathcal{H}}\left[\left\|\sum_{u=1}^{\lambda_\zeta}p_u\nabla F\left(w_{u,q}^{S^u}\right)\right\|^2\right]\right.\\ &\left.\underbrace{+\mathbb{E}_{\mathcal{H}}\left[\left\|\nabla f(w_i)-\sum_{u=1}^{\lambda_\zeta}p_u\nabla F_u\left(w_{u,q}^{S^u}\right)\right\|^2\right]}_{T_3}\right)\end{aligned} \tag{13}$$

where H is the history of communication rounds.

It is worth noting that in the following equations, we utilize the symbol q to denote the number of local epochs, whereas the symbol Q refers to the overall number of local epochs, in order to differentiate between the various terms.

Bound $\mathbb{E}[T_3]$. By adding a zero term combined with $\pm\nabla F_i\left(w^{S^i}\right)$ and relying on the L-smoothness assumption, $E[T_3]$ can be converted to

$$\begin{aligned}\mathbb{E}[T_3]\leq 2\sum_{i=1}^{\lambda_\zeta}p_i^2\sum_{u=1}^{\lambda_\zeta}&\left(L^2\mathbb{E}_{\mathcal{H}}\left\|w_i-w^{S^u}\right\|^2\right.\\ &\left.+L^2\mathbb{E}_{\mathcal{H}}\left\|w^{S^u}-w_{u,q}^{S^u}\right\|^2\right)\end{aligned} \tag{14}$$

Following the Eq. (16) of FedBuff [4], we have

$$\mathbb{E}_{\mathcal{H}}\left\|w^{S^u}-w_{u,q}^{S^u}\right\|^2\leq 3\ q\left(\sum_{q=0}^{q-1}\left(\eta^{(q)}\right)^2\right)\left(\sigma_\ell^2+\sigma_g^2+G\right) \tag{15}$$

Based on Eq. (14) and Eq. (15) of FedBuff [4], $\mathbb{E}_{\mathcal{H}}\left\|w_i-w^{S^i}\right\|^2$ can be transformed to

$$\begin{aligned}\mathbb{E}_{\mathcal{H}}\left\|w_i-w^{S^i}\right\|^2\leq\ & 3Q\lambda_\zeta^2S_{\max}^2\left(\sum_{q=0}^{Q-1}\left(\eta^{(q)}\right)^2\right)\\ &\times\left(\sigma_\ell^2+\sigma_g^2+G\right)\end{aligned} \tag{16}$$

Combing these two terms Eq. 15 and Eq. 16 into Eq. 14, $\mathbb{E}[T_1]$ can be

$$\begin{aligned}\mathbb{E}[T_1]=&-\frac{k}{2}\sum_{q=0}^{Q-1}\eta^{(q)}\|\nabla f(w_i)\|^2\\ &+3\lambda_\zeta\sum_{u=1}^{\lambda_\zeta}p_i^2L^2Q\sum_{q=0}^{Q-1}\eta^{(q)}\sum_{q=0}^{Q-1}\left(\eta^{(q)}\right)^2\end{aligned}$$

$$\times \left(K^2 S_{\max}^2 + 1\right)\left(\sigma_l^2 + \sigma_g^2 + G\right) \underbrace{- \sum_{q=0}^{Q-1} \frac{k\eta^{(q)}}{2} \mathbb{E}_{\mathcal{H}} \left\| \sum_{u=1}^{\lambda_\zeta} p_u \nabla F_u \left(\omega_u^{S^u}, q\right) \right\|^2}_{T_4} \quad (17)$$

Similarly, after introducing the Poisson distribution of arrival updates into the derivation of $\mathbb{E}[T_2]$, we can obtain the following inequality based on the Eq. (23) in [19].

$$\mathbb{E}[T_2] = \mathbb{E}\left[\frac{L}{2}\left\| \sum_{u\in\lambda_\zeta} \sum_{q=0}^{Q-1} \eta^{(q)} g_k\left(w_{u,q}^{S^u}\right)\right\|^2\right] \leq \frac{L\sum_{u\in\lambda_\zeta} p^k \sum_{q=0}^{Q-1}(\eta^q)^2 \sigma_e^2}{2} + \underbrace{\frac{L\lambda_\zeta^2 Q}{2} \sum_{q=0}^{Q-1}\sum_{u=1}^{\lambda_\zeta} \left(\eta^{(q)}\right)^2 p_u \mathbb{E}_{\mathcal{H}}\left[\left\|\nabla F_u\left(w_{u,q}^{S^u}\right)\right\|^2\right]}_{T_5} \quad (18)$$

To guarantee a training process in which the loss decreases gradually, it is necessary to have $\mathbb{E}[f(\boldsymbol{w}_{i+1})] \leq \mathbb{E}[f(\boldsymbol{w}_i)]$. After inserting our derivations Eq. 17 and Eq. 18 into the original objective Eq. 11, we have $\mathbb{E}[f(\boldsymbol{w}_{i+1})] - \mathbb{E}[f(\boldsymbol{w}_i)] \leq T_4 + T_5$ by ignoring other unchangeable terms. Therefore, under the Poisson process of arrival updates, we have $T_4 + T_5$ equals to

$$= \sum_{q=0}^{Q-1}\sum_{u=1}^{\lambda_\zeta}\left(-\frac{\lambda_\zeta \eta^{(q)}}{2}\left(\sum_{u'=1}^{\lambda_\zeta} p_{u'}^2\right) + \frac{L\lambda_\zeta^2 Q\left(\eta^{(q)}\right)^2}{2} p_u\right) \times \mathbb{E}_{\mathcal{H}}\left\|\nabla F_u\left(w_{u,q}^{S^u}\right)\right\|^2 \leq 0 \quad (19)$$

We can set each inner term to be less than zero to guarantee Eq. 19. This leads to

$$\frac{L\lambda_\zeta^2 Q p_u}{2}\left(\eta^{(q)}\right)^2 - \frac{\lambda_\zeta\left(\sum_{u'=1}^{\lambda_\zeta} p_{u'}^2\right)}{2}\eta^{(q)} \leq 0 \quad (20)$$

We can easily obtain $\eta^{(q)} \leq \frac{\left(\sum_{u'=1}^{\lambda_\zeta} p_{u'}^2\right)}{QL\lambda_\zeta p_u'}$. Thus, as the local learning rate satisfy $\eta^{(0)} \geq \eta^{(1)} \geq \ldots \geq \eta^{(Q)}$, Eq. 20 can be achieved when $\eta^{(0)} \leq \frac{\left(\sum_{u'=1}^{\lambda_\zeta} p_{u'}^2\right)}{QL\lambda_\zeta p_u}$.

We then introduce the obtained Eq. 17 and Eq. 18 to Eq. 11 while maintaining the constraint shown by Eq. 20:

$$\lambda_\zeta \sum_{q=0}^{Q-1}\eta^{(q)}\|\nabla f(w)\|^2 \leqslant 2\left(E[f(w_i)] - \mathbb{E}[f(w_{i+1})]\right) + 6\lambda_\zeta \sum_{u=1}^{\lambda_\zeta} p_u^2 \lambda_\zeta Q \sum_{q=0}^{Q-1}\eta^{(q)}\sum_{q=0}^{Q-1}(\eta^{(q)})^2 \times \left(K^2 S_{\max}^2 + 1\right)\left(\sigma_l^2 + \sigma_g^2 + G\right) + L\sum_{q=0}^{Q-1}\left(\eta^{(q)}\right)^2 \sigma_l^2 \quad (21)$$

Finally, following the common proof schema, we first sum up the communication round i from 1 to T for Eq. 21, then eliminate the coefficient term on the left, and finally divide T on both sides, leading to

$$\frac{1}{T}\sum_{i=0}^{T-1} E\|\nabla f(w)\|^2 \leqslant 2\frac{Ef[(w_0)] - E[f(w^*)]}{\lambda_\zeta \sum_{q=0}^{Q-1}\eta^{(q)}} + 6\sum_{u=1}^{\lambda_\zeta} p_u^2 L^2 \lambda_\zeta \times \sum_{q=0}^{Q-1}\left(\eta^{(q)}\right)^2\left(K^2 S_{\max}^2 + 1\right) \times \left(\sigma_l^2 + \sigma_g^2 + G\right) + L\frac{\sum_{q=0}^{Q-1}\left(\eta^{(q)}\right)^2 \sigma_l^2 S}{\lambda_\zeta \sum_{q=0}^{Q-1}\eta^{(q)}} \quad (22)$$

After setting
(A). $\phi(E) = \sum_{q=1}^{E}\eta^q$, $\psi(E) = \sum_{q=1}^{E}(\eta^q)^2$

(B). $\chi(\lambda_\zeta) = \sum_{u=1}^{\lambda_\zeta} p_u^2$

(C). $\sigma^2 = \lambda_\zeta^2\sigma_l^2 + \lambda_\zeta^2\delta_g^2 + G^2$.,
we obtain the convergence bound.

□

The convergence bound that we have obtained expands upon the one presented in PORT by allowing for dynamic arrival updates for the server. This means that our bound is applicable even when the arrival rate λ_ζ is not constant. However, if the arrival rate λ_ζ is constant, our obtained Theorem Eq. 10 simplifies to the one presented by PORT, where the server aggregates a fixed number of updates consistently.

Corollary 1: We assume a constant learning rate η that meets the constraint Eq. 10 during local updates to demonstrate the impact of dynamic arrival updates on convergence. Then, for a sufficiently large T, we obtain

$$\frac{1}{T}\sum_{i=0}^{T-1}\mathbb{E}\parallel\nabla f(\boldsymbol{w}_i)\parallel^2 \leq \mathcal{O}\left(\frac{(f(\boldsymbol{w}_0) - f(\boldsymbol{w}^*))}{\sqrt{TKr\bar{t}E}}\right) + \mathcal{O}\left(\frac{E\left(Kr\bar{t}\right)^2\Omega^2\sigma^2}{T}\right) + \mathcal{O}\left(\frac{E\sigma^2}{T}\right) + \mathcal{O}\left(\frac{\sigma_l^2}{K\sqrt{TKr\bar{t}E}}\right) \quad (23)$$

where $\sigma^2 = \sigma_l^2 + \delta_g^2 + G^2$.

This corollary offers valuable insights for designing and tuning the SPIN algorithm to achieve optimal performance in various practical settings.

♢ *Trade-off between convergence performance and server aggregation timing.* There exists a delicate balance between the frequency of server aggregation and the convergence performance with the SPIN algorithm. By waiting for a longer time before performing aggregation, the server incorporates more updates, leading to a

faster reduction in the loss. However, this also results in a higher gradient variance, which could negatively impact the convergence performance. On the other hand, performing aggregation more frequently reduces the gradient variance but may lead to a slower decrease in loss. Thus, it is crucial to carefully choose the server aggregation interval $K\bar{t}$ to optimize the trade-off between convergence performance and gradient variance.

♢ *Impact of user arrival rate on convergence performance.* A high user arrival rate λ can potentially lead to more updates being included in the aggregation, contributing to a larger drop in loss. However, this also increases the gradient variance, which could affect the convergence performance adversely. To tackle this challenge, the server must monitor the number of updates it receives within the aggregation time interval $K\bar{t}$ and adjust the value of K accordingly. By maintaining a reasonable rate of user arrivals, the server can effectively balance the trade-off between the rate of loss reduction and gradient variance, ultimately achieving better convergence performance.

These insights can guide the design and implementation of the SPIN algorithm in real-world applications, providing practitioners with a better understanding of the factors influencing its performance and enabling them to make informed decisions when tuning its parameters.

V. IMPLEMENTATION AND PERFORMANCE EVALUATION

A. Implementation

To ensure reproducibility of our experiments, we have implemented SPIN in the PLATO framework,[1] which is designed for reproducible and scalable federated learning research.

1) Pull Updates: On the same physical machine, PLATO emulates clients in conventional FL using UNIX processes. In order to implement pull updates that are asynchronously sent to the server when human users generate new data samples, a naïve strategy is to start one process per client, which simulates the timing of a sequence of events — representing the arrival of new data samples — using a random process, and submits a pull request upon the generation of each event. However, if the total number of clients is large, an equal number of client processes needs to be started even if only a small number of clients may be fine-tuning the model simultaneously, which consumes physical memory unnecessarily. Instead, we opt to generate such a sequence of events *on the server* for all the clients, following a Poisson process with an arrival rate of λ.

Since the server needs to run other essential tasks such as aggregating model updates when the need arises, these events need to be generated concurrently with other tasks running on the server. It is well known that with the Global Interpreter Lock (GIL) mechanism to simplify memory management and ensure thread safety, it is not possible for multiple threads in the same process to execute Python bytecodes concurrently. Instead of using multiple threads, we take advantage of the `asyncio` support since Python 3.6, which provides an asynchronous I/O framework with an event loop to enable efficient concurrency. Upon the generation of events in an `async periodic_task` function, which runs periodically, new push updates will be sent to the clients corresponding to these events.

[1]Available as open source at https://github.com/TL-System/plato

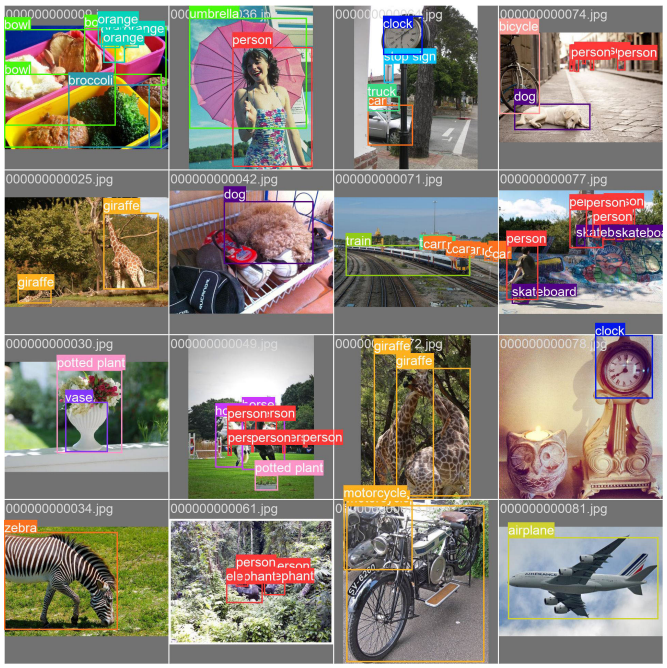


Fig. 3. Fine-tuning the pretrained `yolov8n` model using SPIN.

2) Object Detection With `YOLOv8`: Object detection models are widely deployed in augmented reality applications in the metaverse to identify and track objects in real-time, allowing for the overlay of digital information or graphics onto the real world. These models often need to be fine-tuned with new data samples labeled by human users. Due to the relevance of an object detection task to the metaverse, it should be one of the tasks we use to evaluate SPIN's performance. We chose to implement the state-of-the-art `YOLOv8` model from Ultralytics in PLATO, which outperformed the widely known `YOLOv5` model by a considerable margin.

Unfortunately, the `YOLOv8` trainer provided by Ultralytics does not support customizable samplers in its dataloaders for both `train` and `test` datasets. Samplers are, however, required in PLATO to simulate i.i.d. or non i.i.d. distributions across clients in federated learning. To bridge such a gap, we have implemented a new `SampledDetectionTrainer` class, which inherits from the `DetectionTrainer` class from `YOLOv8` and adds the ability to customize the sampler used by both `train` and `test` dataloaders. The results after fine-tuning the pretrained `yolov8n` model using the `COCO128` dataset and SPIN's implementation in PLATO have been shown in Fig. 3.

B. Performance Evaluation

We are now ready to present a comprehensive evaluation of SPIN across a diverse range of experimental settings.

1) Experimental Settings: Our study involves an array of training sessions involving 100 users and four sets of datasets and models: `MNIST` [20] and `EMNIST` for fine-tuning the `LeNet-5` model, `CIFAR-10` [21] for fine-tuning the `ResNet-18` model, and `COCO128` for fine-tuning the pre-trained `YOLOv8` model as presented previously. The first three sets of benchmark experiments were conducted on a server consisting of 4 NVIDIA RTX A4000 GPUs, using CUDA version 11.6. To best simulate slower client devices in the metaverse, the final set of `YOLOv8` object detection experiments were conducted on a Mac computer running macOS 12.6.3, equipped with an M1 Max CPU and 64GB of unified memory, using the CPU as the fine-tuning device. In conventional FL, data heterogeneity that follow non-i.i.d. data distributions would significantly slow down convergence. In our experiments, when a non-i.i.d. distribution is needed, we employ the Dirichlet distribution with a concentration parameter of 1.

2) Minimum Number of Model Updates K: Using the `MNIST` dataset and the `LeNet-5` model, we begin by conducting a hyperparameter sweep for the minimum number of model updates, K, required before the server initiates aggregation in each round. The elapsed wall-clock time in seconds was used as the critical performance metric until the global model converged. Our experimental findings indicate that, if the number of model updates before aggregation is too small,the converging process will slow down significantly. This can be attributed to the observation that if only a few model updates do not accurately represent the current global state of the model across all the clients, ultimately leading to global model drift in undesirable directions. Conversely, when K becomes large, the system takes significantly longer to converge, with the server waiting for a considerable amount of time to accumulate model updates for each round, leading to a slower overall iteration in the training process. Our experiments revealed that $K = 10$ represents the best trade-off, which is consistent with FedBuff's recommendation [4]. All our subsequent experiments used this value of K.

3) Arrival Rate λ: As the number of clients arriving per unit time varies, SPIN's global convergence rate will be affected due to model staleness, as discussed in Section III. We evaluated the elapsed wall-clock time as the performance metric. Our experimental results, shown in Fig. 4, supported our intuition. As the user arrival rate λ increases, the convergence performance improves, as the impact of staleness is reduced to negligible levels. In 90 seconds, the setting of $\lambda = 0.7$ was the fastest to reach a target accuracy of 96% with the `MNIST` dataset and $K = 10$.

4) Necessity for the Timeout π: In SPIN, in addition to the minimum number of model updates K before aggregation, a timeout π is also imposed to mitigate model staleness. To evaluate the positive effects introduced by incorporating the timeout π, we ran several experiments with varying values of λ, and across three benchmark datasets: the `MNIST` and `EMNIST` datasets, where we employed an `LeNet-5` model, and the `CIFAR-10` dataset, where we employed a `ResNet-18` model. The actual training times have been measured and reflected in our overall wall-clock time elapsed till convergence. Our experimental results are shown in Fig. 5, where we measure the actual training time with and without activating the timeout π in the SPIN algorithm.

Our results clearly demonstrated that SPIN's convergence performance has indeed been improved with the use of the timeout. More specifically, for all three datasets considered in our experiments, with the same setting of λ, the convergence speed with the timeout activated consistently outperformed its counterpart without the timeout, despite the intuition that using the timeout may impair the global model's convergence as fewer model updates will be aggregated. The optimal value of π is determined based on the training time in each task, and our additional experimental results suggested that the optimal timeout value needs to be set at $6/10$ of the average training time consumed in each round. These results also provided further evidence for the impact of λ on the convergence speed of the global model in two additional datasets, over our results previously shown in Fig. 4.

5) Potential Impact of Non-i.i.d. and i.i.d. Data Distributions: In our experimental setup, the clients accumulate data samples as they move through the metaverse, making it difficult to predict whether the generated data is independent and identically distributed (i.i.d.). To avoid any interference from varying data distributions as we evaluate SPIN, we experimented with the same parameter settings on all three datasets but with different data distributions. We hypothesized that non-i.i.d. data distributions — generated with a Dirichlet distribution with a concentration parameter of 1 — would have an adverse effect on the convergence of the global model. The results shown in Fig. 6 partly support our conjecture.

It turned out that our hypothesis is only partially valid with some of the datasets. With the `MNIST` and `EMNIST` datasets, the convergence of the global model followed our hypothesis, as convergence with i.i.d. data consistently underperformed over that with non-i.i.d. data for the same arrival rate. However, with the `CIFAR-10` dataset, non-i.i.d. data actually has a positive effect on convergence, and the global model converges faster with non-i.i.d. data than with i.i.d. data for all values of λ. Though the effects of non-i.i.d. data distributions may not be consistently conclusive across all datasets with the use of SPIN, they are still quite substantial, and should be considered carefully.

6) Overall Performance Evaluations: As SPIN is designed for specialized metaverse scenarios, it is challenging to find closely related work for our comparative study. We chose to compare with FedBuff [4] and PORT [19], the state-of-the-art asynchronous FL mechanism. FedBuff, PORT and SPIN share the same total number of 100 clients and the same settings of λ. In SPIN, a total of 100 clients actively send pull requests to the server after accumulating enough data samples. In contrast, clients in FedBuff and PORT can only be selected by the server. In our experiments, FedBuff and PORT randomly select 20 clients to join the training in each round, and uses the same minimum number of clients $K = 10$ before aggregation; but if a selected client has no data sample generated recently, it cannot participate in the current round. In addition, PORT has a different aggregation algorithm, that is, it will give higher weights to model updates that are similar to the current global

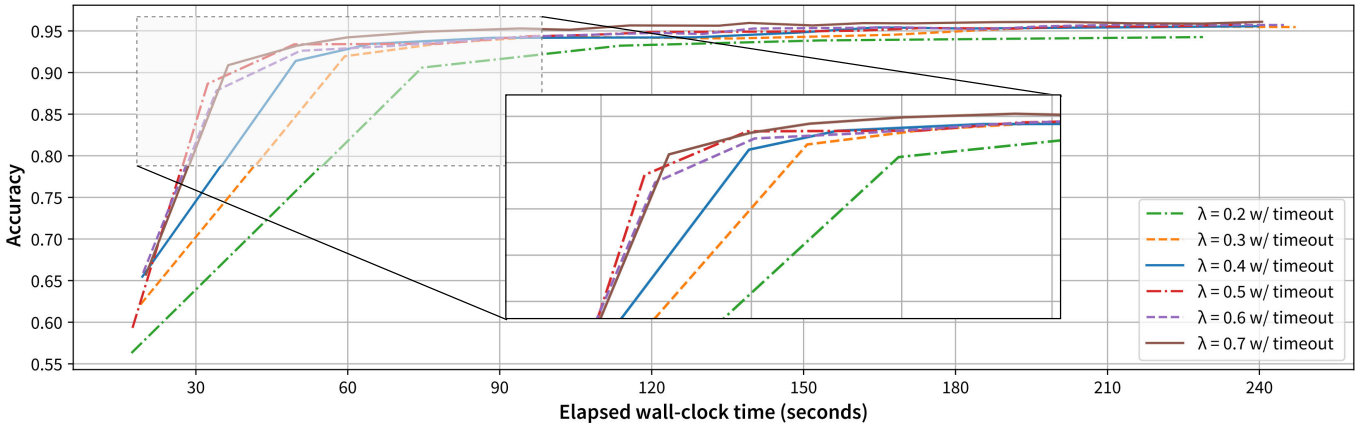
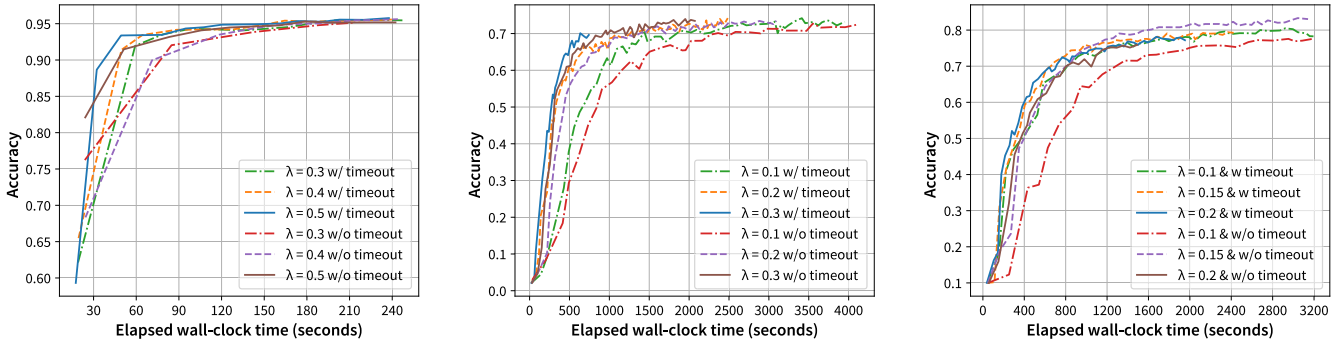
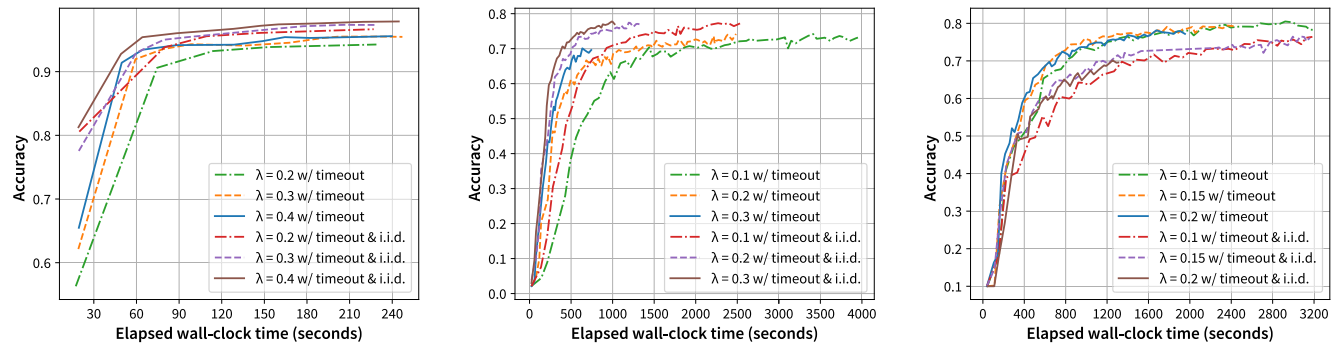


Fig. 4. With the MNIST dataset, we compare the convergence speed of different user arrival rates in SPIN with a timeout π and a mild non-i.i.d. Dirichlet data distribution, with different values of λ, which is the arrival rate used in the Poisson process generating events. Our results confirmed our intuition that higher arrival rates lead to faster the global model convergence, and demonstrated the correctness of our convergence proof in Section IV.



(a) Different λ with & without timeout on MNIST. (b) Different λ with & without timeout on EMNIST. (c) Different λ with & without timeout on CIFAR-10.

Fig. 5. SPIN *with* with different arrival rates λ: a performance comparison with a focus on the timeout π. Data distribution across clients is non-i.i.d.

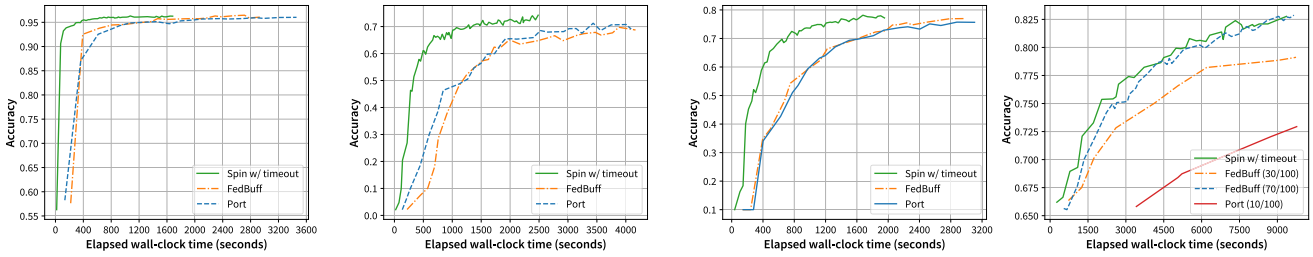


(a) Non-i.i.d. and i.i.d. data distribution with different λ on MNIST. (b) Non-i.i.d. and i.i.d. data distribution with different λ on EMNIST. (c) Non-i.i.d. and i.i.d. data distribution with different λ on CIFAR-10.

Fig. 6. SPIN with timeout π activated and with different values of λ: a performance comparison with a focus on the non-i.i.d. and i.i.d. data distributions.

model when aggregating. PORT also has the related 'timeout' mechanism, which is not measured by time, but by the training state of the stale clients. We conducted additional experiments with FedBuff and PORT to determine the optimal timeout value, as our previous experiments confirmed that setting a timeout is helpful to speed up model convergence.

Our experimental results (Fig. 7) indicated that SPIN significantly outperforms FedBuff. Specifically, for the datasets MNIST, EMNIST, and CIFAR-10, SPIN exhibits a significant advantage. Our experiments evaluate the performance of algorithms by halting upon reaching a prescribed accuracy, calculating the time taken by different algorithms to attain the same accuracy. For the MNIST dataset, SPIN required 962 seconds to converge towards 96% accuracy, whereas FedBuff and PORT took 2290 and 3458 seconds respectively, making SPIN faster by **2.38** times than FedBuff. The gap between these algorithms was also substantial on EMNIST and CIFAR-10. SPIN achieved 72% accuracy on EMNIST and was faster than FedBuff and PORT by factors of **4.11** and **3.30** respectively; it reached 77% accuracy on CIFAR-10,



(a) User arrival rate $\lambda = 0.2$ with timeout on MNIST.
(b) User arrival rate $\lambda = 0.2$ with timeout on EMNIST.
(c) User arrival rate $\lambda = 0.2$ with timeout on CIFAR-10.
(d) User arrival rate $\lambda = 0.01$ with timeout on COCO128.

Fig. 7. SPIN (with timeout π) vs. FedBuff with the MNIST, EMNIST, CIFAR-10, and COCO128, and a non-i.i.d. Dirichlet data distribution.

again out-pacing FedBuff and PORT by factors of **2.06** and **2.07** respectively.

The superior performance of SPIN can be attributed to the setting of an appropriate timeout, which accelerates the FL aggregation process. In other words, within the same timeframe, SPIN completes more rounds than FedBuff and PORT, resulting in faster convergence. The timeout setting is intended to expedite the server's aggregation process. However, if an overly small timeout is set, it would cause a too small number of model updates from the client during each server aggregation, thereby impeding model convergence. Thus, after selecting an appropriate timeout, the performance of SPIN is greatly superior to FedBuff and PORT, which lack a timeout mechanism. Notably, the Push-Pull mechanism in PORT bears some similarity to the timeout in SPIN, as it assists clients with extremely slow training speeds in shortening their training time, preventing them from slowing down the server's aggregation. Moreover, it prevents overly stale model updates from being aggregated, thereby avoiding delaying the convergence of the entire model. However, since there are very few clients with extremely slow training, the performance of PORT is not significant.

In our experiments on the COCO128 dataset with YOLOv8 as the global model and mAP50 (mean Average Precision) as the accuracy metric, we modified the experimental settings to provide a challenging but more realistic environment for our evaluations. For a more realistic setting, we first used only an M1 Max CPU for client fine-tuning rather than NVIDIA GPUs, leading to much longer runs for our experiments. We then set the arrival rate $\lambda = 0.01$, implying that very few events arrive per unit time. We used two settings with FedBuff, where 30% and 70% of all clients were selected in each round. Under the favorable condition of selecting 70% of clients in each round, FedBuff is able to closely match SPIN's performance, but still lags behind by a small margin. Yet, with 30% of clients selected, FedBuff significantly underperformed. To illustrate SPIN's superior performance under equivalent conditions, we subjected PORT to identical circumstances. This entailed a total of 100 clients, with 10 selected per round, and trains YOLOv8 on the COCO128 dataset. Since λ was set to 0.01, the arrival of pull requests was remarkably sparse. If aggregation only depends on the receipt of the minimum number of model updates, it demonstrates a significant under-performance in terms of convergence. The outcome from PORT reinforces this observation. When we set the convergence accuracy target for YOLOv8 at 82%, SPIN achieved the target accuracy in a mere 8015 seconds, whereas PORT required 32723 seconds. Our findings in these experiments, especially with fine-tuning a pre-trained YOLOv8 object detection model, suggested that SPIN is both effective and robust across a wide variety of settings and tasks, in the specialized metaverse scenarios for which it was designed.

VI. RELATED WORK

A. Privacy

Regarding privacy preservation, existing research on enabling users to participate in training machine learning models in the metaverse has explored a variety of approaches. One such method is *differential privacy*, which can be applied to perturb the user data or the model updates during the training process, ensuring that individual user's data or contributions cannot be directly inferred. Differential privacy techniques add carefully calibrated noise to the training process, which provides statistical guarantees on the privacy of individual data points.

An alternative approach is *encrypted computation*, which allows computation to be performed on encrypted data, protecting the privacy of user data throughout the training process. Homomorphic encryption and secure multi-party computation are cryptographic techniques that enable encrypted computation. With homomorphic encryption, data can be encrypted and sent to a central server for computation, while keeping private data confidential. With secure multi-party computation, multiple parties jointly compute a function over their private inputs without revealing the individual inputs, thus preserving privacy.

By applying differential privacy, user data can be protected through the introduction of noise, ensuring that sensitive information is not disclosed. Encrypted computation techniques, such as homomorphic encryption and secure multi-party computation, allow the training process to be performed on encrypted data, preventing any party, including the service provider, from accessing the sensitive information in plaintext. These privacy protection methods can be incorporated into the research on enabling user participation in training machine learning models in the metaverse. They offer mechanisms to preserve privacy while allowing users to contribute their data and participate in the training process, ensuring that sensitive information remains secure and confidential.

TABLE I
THE ELAPSED WALL-CLOCK TIME OF REACHING CONVERGENCE: A COMPARISON

Task	Dataset	Target Accuracy	Speed up by	Elapsed wall clock time		
				SPIN	FedBuff	PORT
Image Classification	MNIST	96%	**3.59**	962s	2290s	3458s
	EMNIST	72%	**4.11**	1794s	7379s	5922s
	CIFAR-10	77%	**2.07**	1558s	3215s	3231s
Object Detection	COCO128	82%	**4.08**	8015s	13191s (30/100)	32723s

B. Incentive Mechanisms

Existing works in the literature have explored incentive mechanisms in the context of federated learning. These mechanisms aim to address the challenge of incentivizing users to contribute their data and computational resources while preserving privacy. One example of such work is Kang et al. [23], which proposed a reputation-based incentive mechanism for federated learning. The authors introduce a reputation-based worker selection scheme for quantifying the reliability and contribution of each user in the federated learning system. This mechanism incentivizes users to actively participate and contribute to the federated learning process.

These studies represented a subset of the existing research on incentive mechanisms in the context of federated learning. By leveraging reputation systems and cooperative game theory, these mechanisms aim to motivate users to actively participate, contribute their data and resources, and ensure the success of federated learning models.

VII. CONCLUDING REMARKS

In this paper, we have presented SPIN, a novel mechanism designed for MLOps in the metaverse that targets fine-tuning globally deployed models that have been pre-trained. We addressed the unique challenges of utilizing human-generated data in a decentralized metaverse environment and proposed a new mechanism that integrates the benefits of conventional FL and the timeliness of continuous integration for sporadically generated data in the metaverse with human interactions over time. By resolving potential conflicts between pull requests from some clients and push updates from others, SPIN improves the convergence performance by considering the staleness of models sent to clients in response to their pull requests. With a mathematically rigorous analysis of its convergence behavior, we proved that convergence is guaranteed with our proposed mechanism.

Our experimental evaluations using image classification and object detection tasks have demonstrated that that SPIN converges efficiently when fine-tuning pretrained models with sporadic and small-sized data generated by human behavior. Our experimental results have also shown that SPIN outperforms FedBuff in terms of convergence performance, which is the state-of-the-art asynchronous federated learning mechanism where the server actively selects clients in each communication round. Our findings suggest that SPIN is both effective and robust in the model fine-tuning tasks we considered in this paper, for which it is specifically designed. Last but not least, we have made both our source code and experimental settings available in the public git repository for the PLATO FL research framework, to ensure the best possible reproducibility of our work.

In this paper, we made the assumption that using new data samples to fine-tune an existing pre-trained model would lead to a better model in terms of accuracy, but we acknowledge that this is not always the case, especially when malicious users intentionally feed low-quality data into the system. There should naturally be some complementary approaches that are designed to established guard rails against such malicious users and low-quality data samples. As our future work, we will study how such malicious users can be excluded from the fine-tuning process, so that data samples of high quality can contribute to improving the quality of the pre-trained models with domain-specific knowledge.

It is worth noting that our general problem formulation is not limited to fine-tuning scenarios in the metaverse, such as object detection tasks with YOLOv8. It is equally applicable to other MLOps scenarios outside of the metaverse, where a pretrained model has already been deployed but still needs fine-tuning, using data obtained from human interaction. A prominent recent example is the recent OpenAI release of the GPT-4 model: it was trained with more human feedback compared to ChatGPT's GPT-3.5, including feedback submitted by ChatGPT users. OpenAI claimed that "continuous improvement from real-world use has been applied to update and improve GPT-4 at a regular cadence as more people use it [23]." As such, we expect that the need for continuously improving deployed models using human interaction will soon become prevalent, both within and outside of the metaverse. As our future work, we will continue to explore new research challenges where models need to be fine-tuned using human interaction, and local training is no longer feasible due to resource constraints.

REFERENCES

[1] M. Xu et al., "A full dive into realizing the edge-enabled metaverse: Visions, enabling technologies, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 656–700, 1st Quart., 2023.

[2] Y. Wang et al., "A survey on metaverse: Fundamentals, security, and privacy," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 319–352, 1st Quart., 2023.

[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Statist. (AISTATS)*, 2017, pp. 1273–1282.

[4] J. Nguyen et al., "Federated learning with buffered asynchronous aggregation," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2022, pp. 3581–3607.

[5] H. Duan, J. Li, S. Fan, Z. Lin, X. Wu, and W. Cai, "Metaverse for social good: A university campus prototype," in *Proc. 29th ACM Int. Conf. Multimedia*, Oct. 2021, pp. 153–161.

[6] J. D. N. Dionisio, W. G. B. Iii, and R. Gilbert, "3D virtual worlds and the metaverse: Current status and future possibilities," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 1–38, Jun. 2013.

[7] Q. Yang, Y. Zhao, H. Huang, Z. Xiong, J. Kang, and Z. Zheng, "Fusing blockchain and AI with metaverse: A survey," *IEEE Open J. Comput. Soc.*, vol. 3, pp. 122–136, 2022.

[8] F.-Y. Wang, R. Qin, X. Wang, and B. Hu, "MetaSocieties in metaverse: MetaEconomics and MetaManagement for MetaEnterprises and MetaCities," *IEEE Trans. Computat. Social Syst.*, vol. 9, no. 1, pp. 2–7, Feb. 2022.

[9] J. Kang et al., "Blockchain-based federated learning for industrial metaverses: Incentive scheme with optimal AoI," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Aug. 2022, pp. 71–78.

[10] M. Xu, D. Niyato, J. Kang, Z. Xiong, C. Miao, and D. I. Kim, "Wireless edge-empowered metaverse: A learning-based incentive mechanism for virtual reality," in *Proc. IEEE Int. Conf. Commun.*, May 2022, pp. 5220–5225.

[11] W. C. Ng, W. Yang Bryan Lim, J. S. Ng, Z. Xiong, D. Niyato, and C. Miao, "Unified resource allocation framework for the edge intelligence-enabled metaverse," in *Proc. IEEE Int. Conf. Commun.*, May 2022, pp. 5214–5219.

[12] Y. Han, D. Niyato, C. Leung, C. Miao, and D. I. Kim, "A dynamic resource allocation framework for synchronizing metaverse with IoT service and data," in *Proc. IEEE Int. Conf. Commun.*, May 2022, pp. 1196–1201.

[13] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "MetaChain: A novel blockchain-based framework for metaverse applications," in *Proc. IEEE 95th Veh. Technol. Conference: (VTC-Spring)*, Jun. 2022, pp. 1–5.

[14] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," in *Proc. NeurIPS Workshop Optim. Mach. Learn. (OPT)*, 2020.

[15] H. Yang, X. Zhang, P. Khanduri, and J. Liu, "Anarchic federated learning," in *Proc. 39th Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 25331–25363.

[16] J. Nguyen, J. Wang, K. Malik, M. Sanjabi, and M. Rabbat, "Where to begin? On the impact of pre-training and initialization in federated learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[17] Y. Tan, G. Long, J. Ma, L. Liu, T. Zhou, and J. Jiang, "Federated learning from pre-trained models: A contrastive learning approach," in *Proc. Neural Inf. Process. Syst. (NeurIPS)*, 2023, pp. 19332–19344.

[18] M. Kamp, J. Fischer, and J. Vreeken, "Federated learning from small datasets," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[19] N. Su and B. Li, "How asynchronous can federated learning be?" in *Proc. IEEE/ACM 30th Int. Symp. Quality Service (IWQoS)*, Jun. 2022, pp. 1–11.

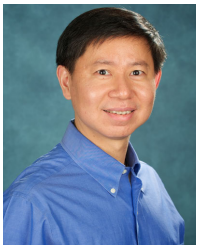
[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Sep. 1998.

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[22] J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang, "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10700–10714, Dec. 2019.

[23] OpenAI. (2023). *GPT-4 is OpenAI's Most Advanced System, Producing Safer and More Useful Responses*. Accessed: Jul. 20, 2023. [Online]. Available: https://openai.com/product/gpt-4

Ningxin Su (Graduate Student Member, IEEE) received the B.E. degree from the Beijing University of Posts and Telecommunications in 2019 and the M.E. degree from the University of Sheffield in 2020. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Toronto. Her research areas include the metaverse, networking, distributed machine learning, and federated learning. She was a recipient of the Best Paper Award from the IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom) in 2023.



Baochun Li (Fellow, IEEE) received the Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana–Champaign, Urbana, in 2000. Since then, he has been with the Department of Electrical and Computer Engineering, University of Toronto, Canada, where he is currently a Professor. He has been the Bell Canada Endowed Chair in computer engineering, since August 2005. His current research interests include cloud computing, security and privacy, distributed machine learning, federated learning, and networking. He is a fellow of the Canadian Academy of Engineering.