Tournament-Based Pretraining to Accelerate Federated Learning

Matt Baughman
University of Chicago
Chicago, IL, United States

Nathaniel Hudson
University of Chicago
Chicago, IL, United States

Ryan Chard
Argonne National Laboratory
Lemont, IL, United States

André Bauer
University of Chicago
Chicago, IL, United States

Ian Foster
University of Chicago
Chicago, IL, United States

Kyle Chard
University of Chicago
Chicago, IL, United States

ABSTRACT

Advances in hardware, proliferation of compute at the edge, and data creation at unprecedented scales have made federated learning (FL) necessary for the next leap forward in pervasive machine learning. For privacy and network reasons, large volumes of data remain stranded on endpoints located in geographically austere (or at least austere network-wise) locations. However, challenges exist to the effective use of these data. To solve the system and functional level challenges, we present an three novel variants of a serverless federated learning framework. We also present tournament-based pretraining, which we demonstrate significantly improves model performance in some experiments. Overall, these extensions to FL and our novel training method enable greater focus on science rather than ML development.

ACM Reference Format:
Matt Baughman, Nathaniel Hudson, Ryan Chard, André Bauer, Ian Foster, and Kyle Chard. 2023. Tournament-Based Pretraining to Accelerate Federated Learning. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3624062.3626089

1 INTRODUCTION

Federated Learning (FL) is a *Machine Learning* (ML) paradigm that enables computing—in this case, training and inference—to be performed where data are generated or live. At its core, FL trains local ML models on some siloed or sharded data and then aggregates those local models into a global model, capturing features of each source of data without directly sharing the raw data. FL solves the problem of data being generated too quickly to transfer or data is restricted due to privacy or other reasons. In some cases [22], data is created at such a scale that not only is it impractical to transfer, it is untenable to store. These cases warrant *online federated learning*, a subclass of FL wherein the dataset is actively changing due to collection and deletion.

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11…$15.00
https://doi.org/10.1145/3624062.3626089

While FL traditionally uses a client-server architecture, with each endpoint requiring independent configuration [5], FL frameworks incorporating serverless as their communication layer to deploy workloads are able to abstract the computation from the hardware. This minimizes or even eliminates the issues associated with tailoring systems level configurations of FL workflows, greatly accelerating usability of HPC and edge devices alike.

In prior work we proposed a serverless FL framework—FLoX [19]—to improve flexible development and usability of diverse device types by abstracting the experiment from the system(s). To support FL exploration at all levels, FLoX is designed to be easy to use and at the same time robust and modular enough to allow for extensive experimentation at any part of the FL process—training individual models, aggregating their weights, distributing the new model, and repeating.

Further challenges for FL arise with data in the wild— FL systems must be robust to data heterogeneity, class imbalances, suboptimal data quality, and system scaling. To explore these challenges and potential solutions for deploying FL workflows in real-world environments, we analyze a largely, unexplored dataset of wildlife images gathered from nearly 200 camera traps. We specifically investigate the use of different FL workflows in this setting. To enable these experiments on more traditional local and HPC resources, we present extensions to the FLoX FL framework [19]. The main contributions of our work are four-fold:

(1) We extend an existing serverless-based FL framework to enable easy execution of FL workflows on remote systems and in HPC environments;
(2) We demonstrate the challenge of scaling FL without the need to modify model architecture or workflow hyperparameters in a real-world application;
(3) We show that data-aware methods do no solve these issues;
(4) We present hybrid tournament-based pretraining as a suitable method to produce better results in certain applications without further system and model-level development.

The remainder of this paper is organized as follows. Section 2 introduces the general problem and the dataset. Section 3 provides an overview of our extensions to FLoX. Section 4 describes the primary FL experimental configurations. Section 5 outlines our novel pretraining method for FL. Section 6 discusses the results as well as their implications on further FL framework requirements, and offers general recommendations for effective FL configuration. Section 8 summarizes our work.

2 DATA DESCRIPTION

We analyze the *Wellington Camera Trap* (WCT) dataset [2], which was collected by 187 cameras located in the suburban and rural areas surrounding Wellington, New Zealand. The images are snapshots of wildlife collected over a period of twenty months by several different camera models. All images have a resolution of 3264×1448 pixels and were taken by motion-activated camera traps, which captured images in 1 to 3-photo bursts. However, the vast majority were 3-photo bursts, which we refer to as sequences. In total, 270,450 images (more than 90,000 sequences) were captured.

Over the course of data collection, 15 unique species (i.e., positives categories) were identified. Two additional categories were created for when no animals were spotted (i.e., a false motion trigger) and when the apparent animal was not identifiable. This labeling was done by expert-trained citizen scientists.

For our experiments, we consider the location and time of each photo as provided in the metadata. Including such data enables recreation of the actual digital environment within which these captures took place, allowing us to simulate *online FL*. Online FL is a critical component of real-world federated learning as it needs to be robust to a changing data environment. In other words, the system may be exposed to more data or forced to discard existing data. As we have the time each photo was taken, we can create a sliding window over the data with respect to time and perform our FL training sequence in a given round just on that data. The performance of such a recreation (both retaining previous data and discarding it) highly indicative of how an FL system may perform if it had been deployed to learn from data in real-time.

We want to highlight that the WCT data presents a particularly challenging classification problem as the intended focus of each image is often obscured, unclear, or in the background (see Fig. 1). For this reason, various ML methods have been unable to achieve high levels of accuracy on the dataset [10].

Figure 1: An example image from the WCT dataset. The trigger of this photo, a bird on the center left, is hard to identify.

2.1 Preprocessing

We downsampled each image to 406 × 306 pixels (a 64× reduction) to be more computationally manageable. Additionally, we omit the unclassifiable class. As one can see in Figure 1, the photo contains meta information and a logo at the bottom. While previous analyses discuss whether to retain or delete this information, we chose to keep this part of the photo. We believe the decision around this cropping to be insignificant to the purpose and results of this analysis. For the experiments, we split the data into training, testing, and validation sets by *sequence*. This split was one at 88%–6%–6% by sequence to ensure representation of each class in each split, given we stratified the division along classes.

2.2 WCT Labels and Data Distribution

To better understand the data, we examine the distribution of the different classes within the data. The most common class is *birds*, making up more than 57% of the dataset. The next most prevalent class is false positives, with nearly 17%, which we omit. The smallest class is *mustelids* (the weasel family), with only 27 images out of 270,450. The exact prevalence of each class is stated in Table 1.

Table 1: Percentage of each class in the training data.

Class	Number	Percentage
Bird	154525	57.14%
Nothing	45721	16.91%
Cat	30285	11.20%
Hedgehog	10706	3.96%
Mouse	6481	2.40%
Rabbit	5714	2.11%
Possum	4351	1.61%
Unclassifiable	3484	1.29%
Ship rat	3183	1.18%
Rat	2757	1.02%
Dog	998	0.37%
Norway rat	861	0.32%
Goat	645	0.24%
Deer	312	0.16%
Hare	207	0.08%
Pig	193	0.07%
Mustelid	27	<0.01%

The 187 camera traps do not have any local compute resources or network connectivity. Therefore, the camera traps were left for a period of time to collect images and then were manually recovered.

Across these 187 sites, a wide variety in quantity of data was collected. The most *popular* camera took a total of 8,831 images, whereas the least popular only provided 2 images to the dataset. The duration of data collection was also not correlated to the number of images provided by a camera. Therefore, it is reasonable to conclude the data produced represents more and less active locations, demonstrating another level of heterogeneity in the data and further motivating data-aware FL.

3 SYSTEM DESCRIPTION

The rapid adoption of FL has spawned a wide variety of FL frameworks serving different needs, workloads, applications, deployment strategies, and hardware configurations. In this work, we build on Flox and extend it to address our target use case. Although we have not performed rigorous system-level performance tests in this paper, we believe that exploring and distilling a functional form of an FL framework is relevant and useful for understanding these experiments. Further, we intend for this detailing to motivate the further design of state-of-the-art, research-oriented FL frameworks.

3.1 FLoX

We build on the FLoX (**F**ederated **L**earning on func**X**) framework [19] (open-source and available on PyPi and GitHub) which provides a simple interface to define FL applications and coordinates execution of FL functions (i.e., training, aggregation, and inference) over a distributed serverless computing framework.

FLoX centers around registering the most compute-intensive functions in a standard FL workflow (e.g., the client-level training function, aggregation, and data management) as invokable serverless functions. This enables function-level modularity where each of the registered functions could simply be replaced with a different function by registering the new function at the appropriate point in the FL flow. FLoX provides the ability to use existing serverless endpoints, share registered functions, and easily push different parts of the FL flow to different compute resources. Most importantly, FLoX exists to power modular FL experiments while abstracting any non-experimental element of the FL process. For instance, a FL experiment can be instantiated with a single line of code.

To facilitate these design principles, FLoX is built on top of the funcX [8] federated function-as-a-service framework. funcX was chosen for several reasons. The funcX client is lightweight, enabling FL on any resource capable of running a Python process. funcX endpoints can be deployed on *any* resource and thus enables FLoX to easily deploy FL across local, edge, cloud, and HPC systems.

3.2 Local, HPC, and Remote FL

We selected FLoX to build these experiments on as its modularity and flexibility allow us to add the functionality we need without having to worry about altering the overall flow of a FLoX experiment. We modified the base FLoX framework for three specific use cases and maintain these as three important variants necessary for a successful research FL framework.

Local FL. We removed the serverless wrappers for each of the functions, allowing each *endpoint* in the new flow to run serially. While this implementation would be less useful for deployed FL systems, serial execution of each endpoint is algorithmically equivalent to a traditional parallel FL flow for research and development applications concerned primarily with algorithmic performance rather than equivalence with respect to system hardware performance. A local implementation is necessary for FL experimentation as small changes in experiment flows may need to be validated prior to running large experiments: small experiments may not require the large compute made available by pushing compute remotely, and there are simply times that external compute is not available. Additionally, this option enables initial use without the need to set up a serverless endpoint or the relevant online accounts. We used this version for shorter running tests, as well as algorithm and data processing development and validation. This implementation was used on commodity laptops and workstations.

FL on HPC. The primary difference between this and the main FLoX branch is the use of the HPC resource's distributed filesystem for model storage and communication and the use of the serverless framework's built-in resource providers to allocate workers dynamically across many different nodes. Distributed file systems, while slow by many I/O standards, represent a significant upgrade to moving around hundreds to even more than 1,000 local FL models for aggregation. Additionally, this allows for research of models that fall outside the current `10MB` per-function transfer limit of the serverless backend of FLoX. The use of a more advanced transfer protocol for moving models has been prototyped for FLoX and excels in HPC performance. The use of the serverless endpoint's resource provider allows for configuration of workers per node and allows for autoscaling the number of nodes dependent on a FLoX-level specification of the total number of workers. We used this FLoX variant for the balanced FL scaling studies (see Sec. 4) as it allowed for rapid testing of FL flows at many scales. We deployed this system on Argonne Leadership Computing Facility's Theta, ThetaGPU, and Polaris HPC systems as well as the University of Chicago Research Computing Center's Midway2 and Midway3 systems.

Fully Remote FL. This variant of FLoX is a hybrid between the two above methods intended to run on a single cloud or remote endpoint but at a larger scale than the local implementation. Therefore, the remote variant also uses the local node storage for models, but all of the functions are placed remotely over serverless, returning a `completed` (or error) message and the location to the stored local model. In this case, aggregation is also placed remotely rather than locally by default. We used this method for all of the tests recreating the original camera trap environment, which were too large to run locally but did not benefit nearly as much or as regularly from scaling out. Addressing the inherent scaling challenges in this paper would hinder a pure examination of data-aware aggregation and is addressed in our previous work. We predominantly used this variant on the Jetstream2 cloud [14] and UChicago Computer Science department workstations.

While these variants are not currently included within the primary FLoX repository, they will be made publicly available for reproducibility and general use among the community, along with the data collected throughout the experiments.

4 EXPERIMENTS

As we seek to investigate the effects of different aggregation methods on global model performance in a real-world FL environment, we structured our experiments to consider multiple forms and scales of data distribution as well as multiple aggregation weighting schemes. For the sake of clarity, we describe each method of data distribution and each method of model aggregation. For all non-centralized data distributions, we test each aggregation method. Finally, we describe how we simulated an online FL use case by progressively performing FL rounds using more and more data chronologically from the data collection process.

In all experiments, we use the MobileNet-V2 [23] architecture as it is the most performant model available and well-validated. For all experiments, we train on the data using a minibatch size of 64 and for a total of 20 epochs. For the non-centralized experiments, we equate this as 5 rounds of 4 epochs per endpoint. For the online FL tests, we performed 5 rounds of training with 20 and 10 epochs per round for releasing and retaining the data (described below), respectively. In this way, each experiment trains for the same total number of data samples.

4.1 Data Configurations

We selected the following data distribution schemes as they represent the primary distributions one may expect to see. These range from centralized data, simulating traditional ML training, to clustered data, simulating performance in increasingly common hierarchical FL scenarios.

Centralized. The typical case of centralizing all data collection and training a single monolithic model.

Even Distribution. In this case, we stratified the data so the distribution of the number of images and representation of the image classes is similar across all endpoints. We did this for 2, 32, 187, and 512 endpoints. This allows us to investigate how performant FL could be under as ideal of data distribution as possible. By performing these tests at multiple scales, we can establish reliable baselines for comparing against the native camera trap distribution.

Recreating Data-Camera Combinations. This distribution is the most critical to our analysis as we distribute data to the endpoints based on the site-ID provided for each image in the dataset metadata. In other words, we effectively recreated the distribution of the data in which it was created. This distribution demonstrates what the performance of our FL-created model would have been if there were co-located compute resources with each of the cameras, effectively simulating how FL would function in a real-world environment.

Clustered Cameras. As hierarchical FL is gaining interest from communities in both research and industry [1], we decided to randomly cluster the dataset by site-ID (as no proximity-indicating location information was available) to simulate how the FL results would change had the cameras had access to smaller, edge-style compute resources that managed or gathered data from a smaller subset of sites. We clustered cameras into groups of 10 and recorded the average performance of three runs for each aggregation method as this test relies on random clustering and the clusters can vary.

Online Federated Learning. We included timing and site information in our simulated recreation of the camera trap data that enables online FL by iteratively exposing data to the system with respect to the time it was collected. Additionally, due to data constraints, remote sensors frequently need to delete old data. Therefore, we enable an optional data cap that discards old data once a specified threshold is reached.

4.2 Aggregation Weighting.

While weighted-FedAvg [6] is one of the simplest FL aggregation algorithms, its versatility makes it a prime candidate for the basis of our experiments. More specifically, weighted-FedAvg performs an averaging function across all local models based on some weighting. This weighting can be calculated based on number of samples, label distribution, or some other factor.

Unweighted. In this case, we weight each model evenly, effectively giving us the standard FedAvg aggregation method (see Eq. 1, where n is the number of endpoints and f is the training function). We use this method to establish an inter-aggregation baseline to see the effect of other weighting methods.

$$w_{t+1}^{global} \leftarrow \frac{1}{n}\sum_{k=1}^{n} f_t^k(w_t^k) \tag{1}$$

On Data Quantity. This is the typical method of weighting for weighted-FedAvg. In this case, we return the number of samples used for training and then apply a softmax across all sample counts to derive a corresponding array of aggregation weights to be used for the respective models (see Eq. 2, where i represents the number of images of a given endpoint).

$$\begin{aligned} \psi &= softmax(I : [i_1, i_2, ..., i_n]) \\ w_{t+1}^{global} &\leftarrow \sum_{k=1}^{n} \psi^k \cdot f_t^k(w_t^k) \end{aligned} \tag{2}$$

On Data Class Representation. We hypothesize that weightings for aggregation can incorporate more data than just sample size to yield more performant resulting models. In this case, we return the number of classes available to each endpoint and take the softmax of that value divided by the total number of classes (see Eq. 3, wherein c is the number of classes represented on a given endpoint). This yields an array of weights corresponding to each contributing model. We hypothesize that this approach will perform better than traditional weighted-FedAvg in circumstances where data may be plentiful but only represents a limited number of classes. This is highly prevalent in the WCT data, given the massive oversampling of bird class images.

$$\begin{aligned} \omega &= softmax(C : [c_1, c_2, ..., c_n]) \\ w_{t+1}^{global} &\leftarrow \sum_{k=1}^{n} \omega^k \cdot f_t^k(w_t^k) \end{aligned} \tag{3}$$

5 TOURNAMENT-BASED PRETRAINING

To account for the sparse features and heterogeneous data distribution, we have developed a technique we called Tournament-Based Pretraining. This method relies on an initial set of training rounds where standard aggregation is forgone in place of model selection based on tournament. We were inspired by the Livermore Tournament Fast Batch (LTFB) algorithm [16] that selects the best performing minibatch update as the new model rather than doing any more complicated gradient update across batches. In their case, this enabled a significant speedup in training.

In our implementation, we execute the first entire round of federated learning using this tournament based method. After each endpoint has completed its respective training regime for the round, local model weights are returned centrally and evaluated against a validation set of data. The set of local model weights that perform

best on this validation test are then distributed as the global model, which will be used to initiate the next training round.

While LTFB was first developed and is used for high rates of weight exchange in FL workflows (i.e., between minibatches rather than following several epochs), we selected it to provide an initial alignment of learned features across all FL clients rather than allowing those features to be averaged away in initial training. While this method could be used in conjunction with any of the above aggregation methods, we chose to use it separately for our experiments so as to demonstrate its potential usefulness.

6 RESULTS

Overall, the experiments of this paper demonstrated that preeminent federated learning approaches are particularly poorly suited to handling a real-world dataset and environment like the Wellington Camera Trap. Tab. 2 displays the results of our tests that divide the data in a balanced matter and those that simulate the original WCT environment. While training the model centrally performs relatively well, naive federated learning does not adequately learn the features of the dataset at each round. The results for other weighting methods are strikingly similar.

However, we notice that with a single round of tournament-based pretraining, certain experiments achieve substantially higher accuracy, reducing observed error by more than 45% in some cases. While the accuracy achieved using pretraining does not approach the centralized accuracy, with 32 endpoints, the accuracy does achieve at least “best guess" accuracy by classifying as a bird every time. In most machine learning, this would be an exceedingly poor result. However, given that naive FL approaches yield no apparent learned features, the emergence in this case of selecting the largest class demonstrates pretraining’s ability to align similar features in client models and to persist those features across multiple rounds. Nevertheless, this benefit breaks down at 187 endpoints.

If we contextualize the results seen in this case against the endpoint and data quantity scaling observations made in Baughman et al. [3], it is likely these experiments have not begun convergence given the sparsity of features and highly imbalanced data. Nevertheless, tournament-based pretraining clearly offers a method to achieve significant accuracy jumps earlier in the training regime and hence motivates its use and further interrogation for application under various training regimes.

Table 2: All primary results from experiments detailed in Sec. 4.

Data Distribution	Unweighted	w/ Pretraining
Centralized	76.0%	-
Even - 2 EPs	4.7%	7.6%
Even - 32 EPs	11.3%	57.9%
Even - 187 EPs	11.3%	1.0%
Even - 512 EPs	11.3%	-
WCT Sites	11.3%	-
WCT Site Clusters	11.4%	-

6.1 Discussion

We ran all experiments multiple times to ensure the relatively stagnant values seen for some experiments in Fig. 2 were not outliers. We hypothesize that the selected model architecture, experimental parameters, and dataset led to natural equilibria when naively aggregating diverse weights. While the experiments performed with pretraining (see Fig. 3) seem to show similar equilibria, the validation accuracies are far more variable round to round than seen without pretraining. This is the most important result of this paper—pretraining led to *increased feature learning and retention* without any changes to model, experimental configuration, or data.

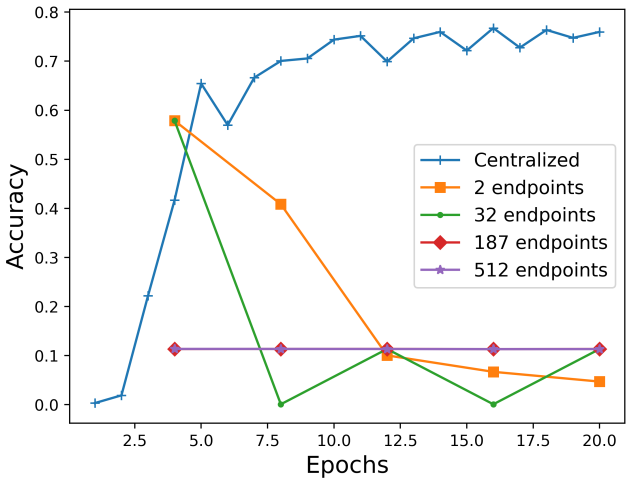

Figure 2: Progression of training for the evenly distributed data against the centralized baseline.

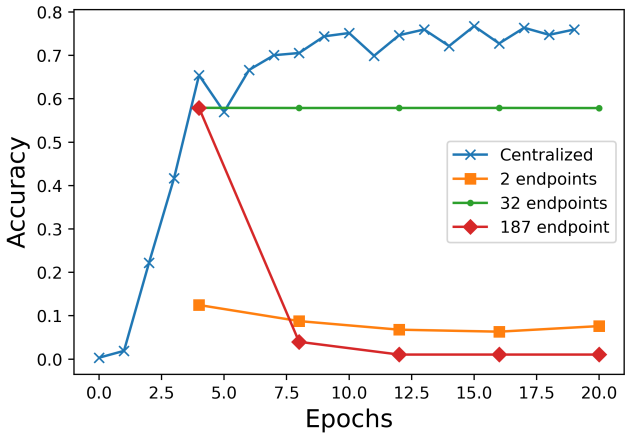

Figure 3: Progression of training when substituting the first round with tournament-based pretraining.

These findings demonstrate the significant need for more advanced but equally as flexible algorithms to enable FL in real-world contexts. Previous work using the same underlying algorithms [19]

demonstrate that these methods work very well for simple image recognition tasks and we can see that the model architecture is capable of learning on centralized data. Therefore, federated learning requires novel methods, like tournament-based pretraining, to solve challenges *not* present in traditional deep learning. Additionally, this may indicate that FL-specific model architectures must be developed to account for data distribution.

6.2 System-Level Findings

These experiments demonstrate the need for a robust research FL framework. We trained a total of more than 10,000 models using our three novel FLoX variants, a task that would have been otherwise impossible. Additionally, the flexibility of the framework enabled use to run experiments on more than 10 different machines (i.e., HPC systems, cloud instances, and local resources), allowing us to test a range of hyperparameters, training flows, and system configurations. The rapid prototyping enabled by the three FLoX variants allowed us to test both system and experiment-level performance without need for individual workflow configurations.

The results and insights we gained from these experiments demonstrates emphatically the need for local, HPC, hybrid, and traditional FL environments that are *intentionally designed for experimentation*. Especially for cases like this, robust experimentation is necessary and that can only be accomplished with robust research infrastructure.

7 RELATED WORK

We build on rapidly growing research in federated learning and systems. As we dwelt less on the side of machine learning solutions in lieu of spending more time motivating adequate research infrastructure, it is important here to recognize the previous works we build on across FL, ML, systems, and the data community.

Federated Learning. There are many FL frameworks available, each with advantages and disadvantages. Notable open-source projects include TensorFlow Federated (TFF) [6], LEAF [7], and PySyft [30]. These frameworks are primarily designed for local deployments and provide basic components for distributed setups. However, they lack convenient interfaces for flexible exchange of auxiliary information or customization of the training procedure. Overcoming these shortcomings, there are other projects like λ-FL [17], FedLess [13], Flower [5], FedScale [20], OpenFed [9], XFL [27], and Parrot [25]. λ-FL tackles the problem of idle aggregators in serverless FL aggregation and deploys FL models on Kubernetes clusters. On the other hand, FedLess implements a distributed model using multiple cloud FaaS providers. Flower implements the FL process from end-to-end with a client-server model. FedScale enables testing FL algorithms against different datasets and in different hardware configurations and data distributions. OpenFed offers the implementation across various topologies while facilitating the exchange of auxiliary information among compute nodes. XFL applies different security mechanisms to prevent the leakage of data. Parrot is based on hierarchical aggregation, where the clients' results are aggregated first locally and then aggregated globally.

Camera Trap Datasets. Camera traps serve various purposes, including biodiversity inventories, assessment of human impact on ecosystems, and more [29]. One of the most popular camera trap datasets is the Snapshot Serengeti [24], which contains 7.1 million images comprising 61 categories. Other data sets cover different regional areas or species [4, 15, 18, 26]. In contrast to these data sets, the WCT data is largely unexplored.

Online FL. Unlike static FL settings, the online setting lacks access to pre-existing data and instead receives data in a streaming fashion, introducing uncertainty and overhead to the system. To tackle these challenges, different approaches such as FedOMD [21], Fleet [11], FedOComp [28], or PSO-Fed [12] have been introduced. FedOMD incorporates multiple local processing steps for communication efficiency and introduces probability risk bounds for generalizability. Fleet is designed to act as middleware between mobile devices and the centralized model to profile and control energy consumption. FedOComp leverages the correlations of the gradients to compress them for over-the-air aggregation, reducing communications overheads. PSO-Fed allows clients to update their local models and share only specific portions of the updated models with the server, granting non-participant clients the opportunity to update their models during a global iteration.

8 CONCLUSIONS

We demonstrated the application of FL in a real-world use-case of camera trap data analysis. The Wellington Camera Trap dataset presents an extreme challenge given the relative difficulty of the images to classify as well as the massively imbalanced data classes. To explore these challenges, we built a set of FL extensions capable of enabling rapid FL experimentation without the need for advanced configurations on top of FLoX, a serverless FL framework. While robust infrastructure does not necessarily lead directly to excellent science results, it does accelerate the rate at which hypotheses can be tested, solutions investigated, and resources efficiently utilized in pursuit of science. Using these variants, we trained more than 10,000 individual models across 10 different compute resources to investigate using FL on camera trap data. Following the demonstration of poor results by traditional and naive FL approaches, we developed and presented tournament-base pretraining. This novel approach demonstrated significant model advantage in some situations and increase model variability across rounds in all cases, as opposed to producing settled equilibria.

Overall, our work offers new tools to the science community to better experiment with FL, illustrates the use of those at rapidly iterating through experiments in a new use case, and demonstrates the benefits yielded by a hybridized training approach, enabling end-user focus on domain science rather than ML configuration.

ACKNOWLEDGMENTS

This research was supported in part by DOE contract DE-AC02-06CH11357 and by NSF grants 1816611, 2004894, and 1550588. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work was completed in part with resources provided by the University of Chicago's Research Computing Center. This work used Jetstream2 at Indiana University through allocation CCR180005 from the ACCESS program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

REFERENCES

[1] Osama Almurshed, Panos Patros, Victoria Huang, Michael Mayo, Melanie Ooi, Ryan Chard, Kyle Chard, Omer Rana, Harshaan Nagra, Matt Baughman, et al. 2022. Adaptive edge-cloud environments for rural AI. In *2022 IEEE International Conference on Services Computing (SCC)*. IEEE, 74–83.

[2] Victor Anton, Stephen Hartley, Andre Geldenhuis, and Heiko U Wittmer. 2018. Monitoring the mammalian fauna of urban areas using remote cameras and citizen science. *Journal of Urban Ecology* 4, 1 (2018), juy002.

[3] Matt Baughman, Nathaniel Hudson, Ian Foster, and Kyle Chard. 2023. Balancing Federated Learning Trade-Offs for Heterogeneous Environments. In *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 404–407.

[4] Sara Beery, Grant Van Horn, and Pietro Perona. 2018. Recognition in Terra Incognita. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*. 472–489. https://doi.org/10.1007/978-3-030-01270-0_28

[5] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. 2022. Flower: A Friendly Federated Learning Research Framework. arXiv:2007.14390 [cs.LG]

[6] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *Proceedings of machine learning and systems* 1 (2019), 374–388.

[7] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).

[8] Ryan Chard, Yadu Babuji, Zhuozhao Li, Tyler Skluzacek, Anna Woodard, Ben Blaiszik, Ian Foster, and Kyle Chard. 2020. Funcx: A federated function serving fabric for science. In *Proceedings of the 29th International symposium on high-performance parallel and distributed computing*. 65–76.

[9] Dengsheng Chen, Vince Junkai Tan, Zhilin Lu, Enhua Wu, and Jie Hu. 2023. OpenFed: A comprehensive and versatile open-source federated learning framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5017–5025.

[10] Benjamin Curran, Seyed Mohammad Nekooei, and Gang Chen. 2022. Accurate New Zealand wildlife image classification-deep learning approach. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 632–644.

[11] Georgios Damaskinos, Rachid Guerraoui, Anne-Marie Kermarrec, Vlad Nitu, Rhicheek Patra, and Francois Taiani. 2022. Fleet: Online federated learning via staleness awareness and performance prediction. *ACM Transactions on Intelligent Systems and Technology (TIST)* 13, 5 (2022), 1–30.

[12] Vinay Chakravarthi Gogineni, Stefan Werner, Yih-Fang Huang, and Anthony Kuh. 2022. Communication-Efficient Online Federated Learning Framework for Nonlinear Regression. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5228–5232. https://doi.org/10.1109/ICASSP43922.2022.9746228

[13] Andreas Grafberger, Mohak Chadha, Anshul Jindal, Jianfeng Gu, and Michael Gerndt. 2021. Fedless: Secure and scalable federated learning using serverless computing. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 164–173.

[14] David Y Hancock, Jeremy Fischer, John Michael Lowe, Winona Snapp-Childs, Marlon Pierce, Suresh Marru, J Eric Coulter, Matthew Vaughn, Brian Beck, Nirav Merchant, et al. 2021. Jetstream2: Accelerating cloud computing via Jetstream. In *Practice and Experience in Advanced Research Computing*. 1–8.

[15] Timm Haucke and Volker Steinhage. 2021. Exploiting Depth Information for Wildlife Monitoring. arXiv:2102.05607 [cs.CV]

[16] Sam Adé Jacobs, Nikoli Dryden, Roger Pearce, and Brian Van Essen. 2017. Towards scalable parallel training of deep neural networks. In *Proceedings of the Machine Learning on HPC Environments*. 1–9.

[17] KR Jayaram, Vinod Muthusamy, Gegi Thomas, Ashish Verma, and Marc Purcell. 2022. Lambda FL: Serverless Aggregation For Federated Learning. In *International Workshop on Trustable, Verifiable and Auditable Federated Learning*. 9.

[18] Benjamin Kellenberger, Thor Veen, Eelke Folmer, and Devis Tuia. 2021. 21 000 birds in 4.5 h: efficient large-scale seabird detection with machine learning. *Remote Sensing in Ecology and Conservation* 7, 3 (2021), 445–460.

[19] Nikita Kotsehub, Matt Baughman, Ryan Chard, Nathaniel Hudson, Panos Patros, Omer Rana, Ian Foster, and Kyle Chard. 2022. FLoX: Federated Learning with FaaS at the Edge. In *IEEE 18th International Conference on e-Science (e-Science)*. 11–20. https://doi.org/10.1109/eScience55777.2022.00016

[20] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*. PMLR, 11814–11827.

[21] Aritra Mitra, Hamed Hassani, and George J. Pappas. 2021. Online Federated Learning. In *2021 60th IEEE Conference on Decision and Control (CDC)*. 4083–4090. https://doi.org/10.1109/CDC45484.2021.9683589

[22] Panos Patros, Melanie Ooi, Victoria Huang, Michael Mayo, Chris Anderson, Stephen Burroughs, Matt Baughman, Osama Almurshed, Omer Rana, Ryan Chard, Kyle Chard, and Ian Foster. 2023. Rural AI: Serverless-Powered Federated Learning for Remote Applications. *IEEE Internet Computing* 27, 2 (2023), 28–34. https://doi.org/10.1109/MIC.2022.3202764

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[24] AB Swanson, M Kosmala, CJ Lintott, RJ Simpson, A Smith, and C Packer. 2015. Data from: Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna. https://doi.org/doi:10.5061/dryad.5pt92

[25] Zhenheng Tang, Xiaowen Chu, Ryan Yide Ran, Sunwoo Lee, Shaohuai Shi, Yonggang Zhang, Yuxin Wang, Alex Qiaozhong Liang, Salman Avestimehr, and Chaoyang He. 2023. FedML Parrot: A Scalable Federated Learning System via Heterogeneity-aware Scheduling on Sequential and Hierarchical Training. *arXiv preprint arXiv:2303.01778* (2023).

[26] Juliana Vélez, William McShea, Hila Shamon, Paula J Castiblanco-Camacho, Michael A Tabak, Carl Chalmers, Paul Fergus, and John Fieberg. 2023. An evaluation of platforms for processing camera-trap data using artificial intelligence. *Methods in Ecology and Evolution* 14, 2 (2023), 459–477.

[27] Hong Wang, Yuanzhi Zhou, Chi Zhang, Chen Peng, Mingxia Huang, Yi Liu, and Lintao Zhang. 2023. XFL: A High Performace, Lightweighted Federated Learning Framework. *ArXiv* abs/2302.05076 (2023).

[28] Ye Xue, Liqun Su, and Vincent K. N. Lau. 2022. FedOComp: Two-Timescale Online Gradient Compression for Over-the-Air Federated Learning. *IEEE Internet of Things Journal* 9, 19 (2022), 19330–19345. https://doi.org/10.1109/JIOT.2022.3165268

[29] Stuart Young, Johanna Rode-Margono, and Rajan Amin. 2018. Software to facilitate and streamline camera trap data management: A review. *Ecology and Evolution* 8, 19 (2018), 9947–9957.

[30] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, et al. 2021. PySyft: A library for easy federated learning. *Federated Learning Systems: Towards Next-Generation AI* (2021), 111–139.