Exploring Federated Learning for Thermal Urban Feature Segmentation - A Comparison of Centralized and Decentralized Approaches

Leonhard Duda[1(✉)], Khadijeh Alibabaei[1], Elena Vollmer[2], Leon Klug[2], Valentin Kozlov[1], Lisana Berberi[1], Mishal Benz[1], Rebekka Volk[2], Juan Pedro Gutiérrez Hermosillo Muriedas[1], Markus Götz[1], Judith Sáinz-Pardo Díaz[3], Álvaro López García[3], Frank Schultmann[2], and Achim Streit[1]

[1] Scientific Computing Center (SCC), Karlsruhe Institute of Technology (KIT), Eggenstein-Leopoldshafen, Germany
leonhard.duda@kit.edu
[2] Institute for Industrial Production (IIP), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[3] Instituto de Física de Cantabria (IFCA), CSIC-UC, Avda. los Castros s/n, Santander, Spain

Abstract. Federated Learning (FL) is an approach for training a shared Machine Learning (ML) model with distributed training data and multiple participants. FL allows bypassing limitations of the traditional Centralized Machine Learning (CL) if data cannot be shared or stored centrally due to privacy or technical restrictions – the participants train the model locally with their training data and do not need to share it among the other participants. This paper investigates the practical implementation and effectiveness of FL in a real-world scenario, specifically focusing on unmanned aerial vehicle (UAV)-based thermal images for common thermal feature detection in urban environments. The distributed nature of the data arises naturally and makes it suitable for FL applications, as images captured in two German cities are available. This application presents unique challenges due to non-identical distribution and feature characteristics of data captured at both locations. The study makes several key contributions by evaluating FL algorithms in real deployment scenarios rather than simulation. We compare several FL approaches with a centralized learning baseline across key performance metrics such as model accuracy, training time, communication overhead, and energy usage. This paper also explores various FL workflows, comparing client-controlled workflows and server-controlled workflows. The findings of this work serve as a valuable reference for understanding the practical application and limitations of the FL methods in segmentation tasks in UAV-based imaging.

L. Duda—Code available in: https://github.com/ai4os-hub/thermal-urban-feature-segmenter.git.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15648, pp. 285–302, 2025.
https://doi.org/10.1007/978-3-031-97000-9_18

Keywords: Federated Learning · Distributed Learning · Real-world Implementation · Segmentation · Energy Consumption · Thermal Anomaly Detection

1 Introduction

Deep Learning (DL) has revolutionized many fields in science and industry, enabling significant advances in tasks like object detection and segmentation [3,31]. DL models, fueled by large datasets, have made it possible to develop systems that perform tasks with human-like accuracy. They are now applied across a variety of industries including healthcare, city management, and autonomous driving [4,13,46].

Traditionally, these models are trained with a large amount of data stored in a central location. However, in response to growing data protection laws such as the General Data Protection Regulation (GDPR) and the EU AI act [33], the question of whether the traditional approach of storing and processing all data centrally is sustainable arises. On the other hand, if the training data is distributed across multiple locations and sharing is hindered due to privacy or resource constraints, successful training becomes challenging, as there may not be enough data available at some of these locations, leading to poor model performance or failure to generalize over unseen data.

To achieve the same predictive performance when the data is spread across multiple locations, Federated Learning (FL) was introduced in 2017 [29]. With this approach, the training data does not have to be shared, but a Machine Learning (ML) model can still be trained collaboratively and in a distributed manner. In the initial introduction of FL, there is a central server that aggregates updates of models trained on decentralized devices. Each device trains a model using its local data and sends only the model updates (not the raw data) to the central server. The server then combines local model weights to update the global model weights. This process enables collaborative model training while reducing the need to share raw data; however, information leakage risks may still exist without additional privacy safeguards such as differential privacy [8,45].

The main objective of this paper is to investigate the effectiveness of FL in real-world applications and scenarios, with a particular focus on a segmentation task in unmanned aerial vehicle (UAV)-based imaging. Although FL has shown promising results in research papers, primarily in simulation scenarios, its practical performance in real-world settings should be explored, considering the limitations and challenges associated with real-world implementation [1]. This study aims to bridge this gap by evaluating the effectiveness of FL in a unique real-world use case: the detection of thermal anomalies in urban environments using UAV-based imaging. This application, part of the AI4EOSC project [2], involves detecting thermal anomalies caused by false alarms from urban features which complicates the detection of locations requiring maintenance or improvement. By reducing false positives, the system supports more efficient optimization of energy-related systems [28,42,43].

A key feature of this work is the use of a real-world dataset collected from two cities, Munich (MU) and Karlsruhe (KA), in Germany. This dataset is characterized by significant imbalances and non-IID (non-independent and identically distributed) data across locations, posing unique challenges for both training and aggregation methods in FL.

The main contributions of this paper are as follows:

- Highlight the real-world challenges associated with training FL models on non-IID, imbalanced data from multiple locations using UAV devices, showcasing the uniqueness of the data and its impact on FL performance.
- Introduce and evaluate FL aggregation strategies tailored to handle non-IID data in segmentation tasks on the UAV images.
- Compare the performance of FL and Centralized Learning (`CL`) on a segmentation task in a real-world UAV imaging application, emphasizing practical challenges such as model accuracy, training time, and memory consumption.
- Investigate the performance of decentralized FL as a potential solution for privacy-preserving collaboration, comparing it against centralized FL in a real-world setting.
- Measure energy consumption and convergence time for both FL and `CL`, addressing the resource-constrained devices used in the real-world application and evaluating their feasibility in a practical deployment.
- Explore the challenges and opportunities of using HPC systems as clients in FL, considering their potential to enhance computational capabilities.

As most of the experiments are done in real-world scenarios, this paper can be used as a valuable reference for understanding the practical applications and performance of FL in real-world settings.

The paper is organized as follows: Sect. 2 reviews the related work in the research areas of FL algorithms and workflows. Section 3 covers the methodology and setup, explaining our concrete implementation of a FL workflow based on the use case described, including a comparison and evaluation of all workflows with each other and with the centralized approach. The obtained results of applying different approaches are evaluated and discussed in Sect. 4.

2 Related Work

FL has gained significant traction in domains such as medical imaging and autonomous driving due to stringent privacy requirements [5]. In traditional distributed learning, it is assumed that all data comes from the same distribution, ensuring consistency across all participating nodes. But in real-world FL, data is typically non-IID, meaning that data across different clients can have different distributions, which poses additional challenges for model training and performance [20]. While the FL approach is effective in simulation environments, it poses challenges in real-world applications due to this non-IID nature of the data and privacy concerns [26,44]. To solve these issues, researchers have proposed

improvements to aggregation algorithms in FL such as `FedProx` [24], Federated Optimization (`FedOpt`) [36], `Scaffold` [22], and `FedBN` [25].

FL has been successfully used mostly in domains like medical imaging and autonomous driving primarily due to concerns about data privacy in these fields [9,14]. In autonomous driving, FL is used to train a semantic segmentation model on data from cars in different cities, achieving robust performance [30]. In medical image segmentation, FL has enabled institutions to build combined models that generalize better across patient populations. For example, [27] used a FL model for brain tumor segmentation utilizing data from multiple hospitals enhancing the performance and generalizability of the segmentation. Furthermore, [35] evaluates several FL aggregation strategies, including `FedProx`, `FedMOON` [23], and `FedDC` [21], alongside `FedAvg`, under both IID and non-IID scenarios in Medical Object Detection (MOD) using simulation.

Beyond these established fields, FL has shown promising potential within UAV systems [11]. For example, [10] explored its use in drone-based collaborative learning. By employing a set of drones that communicate over a wireless network, the study successfully trained a U-Net model for collision avoidance and landing assistance, showcasing potential of FL in edge-based, real-time applications.

Decentralized FL approaches such as Decentralized Cyclic Weight Transfer (`DCWT`) and Swarm Learning (`SL`) eliminate the dependency on a central server and shift the administration tasks to one of the clients. `DCWT` follows a sequential training process where model weights are transferred from one client to the next in a predefined order. Each client trains the model locally before passing the updated weights to the next client, forming a cyclic training approach. `SL` on the other hand, lets all the clients train simultaneously with one client being responsible for the administration tasks [45].

Recent attention has also been drawn to the environmental implications of FL. Studies indicate that FL can be more energy-intensive and generate a larger carbon footprint compared to traditional Centralized Learning (`CL`) methods [34]. The carbon emissions in FL is not only related to the training of the model on client devices, but also includes the energy used during client-server communication and the conversion of energy to carbon emissions depending on regional grid characteristics [37]. Qiu, X. et al. [34] showed that communication costs between clients and servers can represent anywhere from 0.7% to more than 96% of total emissions in FL systems. Farsi, A.A. et al. [12] proposed adding sustainability as a fourth pillar to the existing framework, alongside legality, ethics, and robustness. This addition addresses the environmental impacts of FL systems. Despite these concerns, direct comparative analyses focusing on energy efficiency across various FL algorithms remain limited.

This work builds on these foundations by evaluating FL in a real-world scenario of UAV-based thermal anomaly detection. It examines the performance of centralized and decentralized FL workflows and compares their effectiveness in terms of metrics such as model accuracy, energy consumption, and communication overhead. By using real-world data from two cities with significant

heterogeneity, this study aims to bridge the gap between theoretical FL advances and practical applications.

3 Material and Methods

3.1 Dataset

A multispectral image dataset from [39] forms the basis of this study, publicly available on Zenodo [41]. It consists of **793 images** acquired through 14 UAV flights in Germany, the large majority of which stem from Munich, the rest from Karlsruhe. All flights were carried out in nadir (90° pitch angle) and at 60 m height.

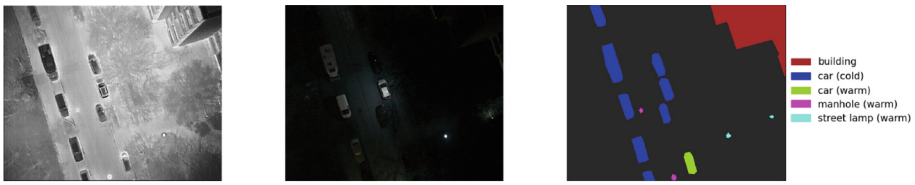
Both thermal infrared (TIR) and standard red-green-blue (RGB) imagery was simultaneously captured using DJI's Zenmuse XT2, a combined 4k RGB camera and FLIR thermal sensor [39]. Registration procedures help compensate for differing aspect ratios, resolutions, and fields of view to create the multispectral dataset [39]. As the focus lies on hot-spot detection, the TIRs were annotated with **seven classes** of common thermal urban features. The classes are summed up in Table 1. An example multispectral image and associated annotation are shown in Fig. 1. Both clearly highlight the strong imbalance of instances and pixel amounts per class [39].

Table 1. Aggregated object counts for KA and MU datasets [39]

General	Client-1 (MU)	Client-2 (KA)	Total (KA+MU)
No. images	700	93	793
Building	1215	189	1404
Car (cold)	838	1694	2532
Car (warm)	86	950	1036
Manhole (cold)	25	495	520
Manhole (warm)	82	1297	1379
Miscellaneous	81	-	81
Person	275	-	275
Street lamp (cold)	5	95	100
Street lamp (warm)	11	672	683

3.2 Thermal Urban Feature Semantic Segmentation

During the detection of thermal anomalies in UAV images, the majority of false positives arise from the common urban features listed in Table 1. The U-Net model is employed to identify these features and eliminate them, thus avoiding their misclassification as potential district heating network leaks [39]. U-Net is a



(a) TIR image (b) RGB image (c) Annotation mask

Fig. 1. Multispectral (RGB/TIR) image and annotations example [39].

convolutional neural network architecture designed especially for image segmentation tasks [15]. Its distinctive U-shaped structure consists of the main parts of an encoder and a decoder, connected by skip connections. The encoder, or contracting path, follows the typical architecture of a convolutional network, while the decoder recovers spatial information and generates the predicted segmentation mask. In this work, a ResNet-152 [17] encoder was used, also referred to as the backbone of the U-Net.

3.3 Framework

In order to apply the FL methods and algorithms and to transform an existing centralized ML workflow into a FL one, we use NVIDIA FL Application Runtime Environment (NVFlare) [18] in this work.

NVFlare is an open source framework for implementing FL workflows in Python. Apart from enabling the transformation of existing ML/DL workflows into a FL paradigm, it offers a wide range of features, which include: simulation and prototyping tools, privacy-preserving algorithms, built-in FL algorithms, productivity enhancement (like MLFlow [32]), provisioning tools for participant verification and secure communication, and many more [18].

3.4 Aggregation Algorithms and Workflows

The most commonly used aggregation algorithms in FL, including `FedAvg`, `FedProx`, `FedOpt`, and `Scaffold`, were selected for this work. `FedAvg` is the first and simplest algorithm introduced in FL, with others developed to address its drawbacks regarding the non-IID nature of datasets in FL settings.

As mentioned before, `FedAvg` uses weighted averages when aggregating local weights sent back from the clients.

`FedProx` introduces a proximal term $\frac{\mu}{2}\|w - w^t\|^2$ to the local objective function, where w is the local model weights, w^t is the global model weights, and μ controls regularization.

`FedOpt` uses stochastic gradient descent (SGD) to update the global model. The global gradient update is:

$$\Delta_t = \frac{1}{|S|}\sum_{i \in S}(w_{i,K}^t - w^t)$$

where S is the set of participating clients, $w_{i,K}^t$ is the local model after K training steps, and w^t is the global model weights in round t. The global model is then updated as:

$$w^{t+1} = \text{ServerOpt}(w^t, -\Delta^t, \eta, t)$$

where η is the server learning rate and ServerOpt refers to the optimization function or algorithm applied by the server to update the global model.

`Scaffold` is an enhanced FL algorithm that mitigates client drift caused by data heterogeneity using control variates. The global update is given by:

$$w^{t+1} = w^t + \eta \frac{1}{|S|} \sum_{i \in S} (\Delta_i^t - c_i^t + c^t), \quad c^t = \frac{1}{N} \sum_{i=1}^{N} c_i^t \tag{1}$$

where w^t is the global model in round t, Δ_i^t the local update, c_i^t and c^t are client and global control variates, η the learning rate, and $|S|$ the number of participating clients.

When FL was first introduced, it initially consisted of a Scatter & Gather (S&G) workflow, relying on a central server for aggregation.

When there is no trusted server available, Decentralized Federated Learning (DFL) such as Decentralized `CWT` (`DCWT`) and Swarm Learning (`SL`) can be used as alternatives. In such cases, communication is conducted peer-to-peer. Since there is no server to be compromised, DFL provides enhanced privacy protection compared to centralized approaches and it reduces network communication. In `SL`, one of the clients takes responsibility for performing aggregation. In the implementation of these workflows in NVFlare, a server is still present to handle administrative tasks, but no sensitive data is shared with it.

3.5 Training Setup

Training the semantic segmentation model requires considerable computational resources, so each client was assigned a GPU node on a HPC system. **Client-1** utilized a GPU node on HoreKa [19], while **Client-2** used a GPU node on HAICORE [16]. Both nodes provide the same resources, including `2 Intel Xeon Platinum 8368` processors with a combined `76 CPU Cores` and `152 CPU Threads`, `512 GB main memory`, `4 NVIDIA A100-40 GPUs` with `40 GB Memory` each and a local `NVMe SSD disk` of `960 GB size`. However, only one GPU from the four GPUs available on each node was used for each client. To run NVFlare clients on our HPC cluster, we used the SLURM job scheduler [38]. Using NVFlare's provisioning function several bash scripts for launching clients and the server are created. We modified these scripts to make them compatible with SLURM, enabling deployment on the HPC systems.

Since our dataset consists of images from two cities, our initial configuration was a simple location-based split with one client per city, as shown in Table 1. To evaluate the scalability another setup including five clients was tested. For

this the data was splitted into five heterogeneous datasets to measure the performance of the FL algorithms with an increased number of clients while maintaining the same overall dataset size. In this setup, due to resource constraints, four clients were launched as a SLURM job, each running on the same node with a dedicated GPU. The fifth client was started with a separate SLURM job to prevent GPU memory allocation issues during training. Table 2 shows the object count and number of images per client. Still, the class count varies across the clients, and this significant variance indicates that the underlying distributions are not identical.

Table 2. Object counts for 5-clients

General	Client-1	Client-2	Client-3	Client-4	Client-5
No. images	159	159	160	164	151
Building	181	459	294	213	257
Car (cold)	231	1093	303	165	740
Car (warm)	72	123	181	302	358
Manhole (cold)	135	74	80	144	87
Manhole (warm)	367	166	293	287	266
Miscellaneous	135	74	80	144	87
Person	0	0	85	137	53
Street lamp (cold)	45	6	19	13	17
Street lamp (warm)	118	131	159	145	130

The server was set up on the bwCloud provided by KIT. The server ran as a Virtual Machine (VM) with `32 GB main memory` and `40 GB storage` [7].

Before sending a model to production, it is important to track and monitor it in order to optimize and check its quality. To this end, MLOps is an engineering practice that aims to automate and streamline the ML lifecycle. For this purpose, in this work MLflow was chosen for experiment tracking as it is supported by NVFlare, and an MLflow tracking server in production was provided by the AI4EOSC project [2]. During training, each client sends local metrics and parameters to the NVFlare server, which logs them on the configured MLflow server. By default, NVFlare does not log the global model in MLflow, so we modified NVFlare source code to enable this functionality for each experiment.

The metrics chosen for evaluating the model consist of the mean accuracy (mACC), mean weighted precision (mwP), mean weighted F_1-score (mwF_1) and mean weighted Intersection Over Union score (mwIoU). The weights in these metrics are determined by the number of pixels occupied by each specific class within the annotation masks.

The thermal urban feature semantic segmentation use case, originally developed and trained in a centralized way with predefined hyperparameters, was adapted into a FL workflow. The training hyperparameters identified by [40]

were already optimized and considered ideal for this application. The parameters for the FL workflow were aligned with those of the `CL` workflow. The sigmoid focal cross-entropy loss function, Adam optimizer, a learning rate (lr) of 0.001 were used [39,40]. Experiments used 2 or 5 clients, with 4 rounds for two clients and 4, 7, or 13 for five clients. Local epochs were 18 for `CL`, 5 or 9 for five clients, and batch size was fixed at 8. As is [39], the dataset was split into 80% training and 20% testing. As shown in Table 2, the dataset is imbalanced which is addressed by the authors in greater detail in [39]—who apply techniques such as transfer learning and tailored loss functions and class-balanced metrics (e.g., weighted precision, F_1, IoU). Since we used the same model and configuration as in [39], the data imbalance is already addressed.

To efficiently calculate the energy consumption of each algorithm, we utilized the *perun* package to track the power draw of CPU, GPU, and memory of both clients and server [15]. Since our original server was set up on a VM, we could not efficiently track the energy consumption due to the lack of direct access to hardware performance counters. To overcome this limitation, we set up the server on bare-metal equipped with hardware energy measurement capabilities (such as Intel RAPL or a similar power monitoring framework) on a dedicated laptop. This setup allowed us to obtain more accurate energy consumption data for our algorithms and compare them.

4 Results and Discussions

The model is trained using the available images to automatically detect and eliminate false alarms. The script for this workflow is written in Python 3.8 and uses Tensorflow 2.10. The results of training multiple models using the various approaches introduced in Sect. 3.4 are presented and discussed in this section.

4.1 Comparison of the Aggregation Algorithms

This section presents the results of applying different FL aggregation algorithms, specifically `FedAvg`, `FedProx`, `FedOpt`, and `Scaffold`. Each experiment was conducted five times to ensure statistical significance. We first trained the model with two clients (KA and MU dataset). This experiment is referred to as the *2-client scenario*. Then, the dataset was divided into five equivalent subsets, and the experiment was repeated to investigate the results when all clients had datasets of equal size within the *5-client scenario*.

In `FedOpt` and `Scaffold`, where server-side optimization updates trainable weights, performance dropped after two rounds and continued to decline with each subsequent round when using standard Batch Normalization (BN) layers [6] in the U-Net backbone, despite weighted averaging of non-trainable parameters like running mean and variance. In BN, locally updated running statistics vary across clients, leading to inconsistencies that degrade global model performance and stability when aggregated. To address this, we replaced BN with Group Normalization (GN) [47], which normalizes within channel groups instead of batches,

making it independent of batch size and more robust for FL. Moreover, since `FedOpt` uses SGD to update the global model, `lr` and `momentum` can be tuned to optimize the update. NVFlare uses default `lr` 1.0 and `momentum` of 0.9, and our experiments showed that reducing `lr` below 1.0 leads to a performance drop in the global model performance. We therefore adhered to the default values.

Table 3 shows the results of training the model within the *2-client scenario*. The first number reported for each client represents the average metric value over five executions of the algorithm, while the second number denotes the standard uncertainty. For the overall results, we computed a weighted average of each metric from both Client-1 and Client-2.

The global model aggregated using `FedAvg` achieved the highest performance across all metrics on the local test set of Client-1 (MU). It recorded a mean weighted precision of 0.945 ± 0.003, a mean accuracy of 0.94 ± 0.007, and an F_1 score of 0.937 ± 0.005. On the other hand, `FedProx` showed the weakest performance for this client, particularly in terms of mean accuracy 0.92 ± 0.01 and mean weighted IoU (0.85 ± 0.02). When comparing `FedAvg` with `Scaffold`, `FedAvg` shows slightly better performance in terms of mACC, mwF_1, and mwIoU. `FedOpt` consistently shows the weakest performance for Client-1, with values that often fall below other ranges of algorithms.

The global model aggregated using `Scaffold` demonstrated the best performance among FL algorithms on Client-2 and has the closest performance to `CL`. It surpasses `FedAvg` in both mACC (0.846 ± 0.009 vs. 0.66 ± 0.007) and mwF_1 (0.858 ± 0.007 vs. 0.810 ± 0.014). Additionally, `Scaffold` mwIoU of 0.777 ± 0.009 shows an improvement over `FedAvg` 0.737 ± 0.014. `FedProx` consistently shows the weakest performance for Client-2, with the widest error ranges indicating less stability.

When comparing `CL` with FL approaches based on overall results, the model trained using the `CL` method and the global model aggregated using `FedAvg` and `Scaffold` show comparable performance across several key metrics. For instance, `FedAvg` achieves a mwP of 0.935 ± 0.005, which is statistically consistent with `CL` with mwP of 0.939 ± 0.002. These results highlight that `FedAvg` and `Scaffold` can achieve performance comparable to centralized training while preserving data privacy. When considering error ranges, `Scaffold` and `FedAvg` can match or even exceed `CL` in specific metrics. Overall, this highlights their potential as effective FL strategies for our application. In contrast, `FedProx` and `FedOpt` show lower performance across several metrics, suggesting that while they can be effective, they may be more sensitive to client data variability.

In the subsequent experiments, we used the perun package [15] to measure the energy consumption of each FL algorithm during the training process. It also offers a calculation of an average carbon intensity of electricity based on a emissions factor which can differ per region and can be set manually by the user. This calculation was not reported within this work due to being highly versatile between different regions of the world. To ensure the reliability and consistency of our results, we took specific precautions to eliminate potential interference. We requested exclusive access to the HPC nodes where our clients were executed,

Table 3. Performance of the global model trained with different FL Algorithms and `CL` on the test set of Client-1 (MU), Client-2 (KA), and (MU+KA)

	Metric	Centralized	FedAvg	FedProx	FedOpt	Scaffold
Client-1	mwP	$\mathbf{0.952 \pm 0.017}$	0.945 ± 0.003	0.924 ± 0.007	0.913 ± 0.001	0.938 ± 0.001
	mACC	$\mathbf{0.949 \pm 0.003}$	0.940 ± 0.007	0.911 ± 0.006	0.879 ± 0.009	0.923 ± 0.004
	mwF_1	$\mathbf{0.944 \pm 0.002}$	0.937 ± 0.005	0.903 ± 0.009	0.887 ± 0.005	0.924 ± 0.002
	mwIoU	$\mathbf{0.907 \pm 0.005}$	0.900 ± 0.007	0.850 ± 0.01	0.822 ± 0.009	0.877 ± 0.004
Client-2	mwP	0.870 ± 0.015	0.853 ± 0.012	0.834 ± 0.025	0.843 ± 0.005	$\mathbf{0.879 \pm 0.003}$
	mACC	0.866 ± 0.012	0.846 ± 0.009	0.818 ± 0.017	0.833 ± 0.005	$\mathbf{0.866 \pm 0.007}$
	mwF_1	0.847 ± 0.017	0.810 ± 0.014	0.779 ± 0.031	0.818 ± 0.004	$\mathbf{0.858 \pm 0.007}$
	mwIoU	0.774 ± 0.018	0.737 ± 0.014	0.702 ± 0.029	0.733 ± 0.005	$\mathbf{0.777 \pm 0.009}$
Overall	mwP	$\mathbf{0.939 \pm 0.002}$	0.935 ± 0.005	0.913 ± 0.011	0.905 ± 0.002	0.931 ± 0.001
	mACC	$\mathbf{0.931 \pm 0.002}$	0.930 ± 0.007	0.900 ± 0.008	0.873 ± 0.008	0.917 ± 0.004
	mwF_1	$\mathbf{0.928 \pm 0.002}$	0.922 ± 0.007	0.888 ± 0.013	0.879 ± 0.006	0.916 ± 0.003
	mwIoU	0.884 ± 0.003	0.881 ± 0.008	0.833 ± 0.014	0.812 ± 0.009	$\mathbf{0.865 \pm 0.005}$

ensuring that no other jobs were running on the same nodes during the experiments. This isolation minimized resource contention and provided a controlled environment for energy measurement. In the *5-client scenario*, four clients shared the same node, making it challenging to measure the energy consumption of each client individually. Therefore, we conducted energy consumption measurements for only two clients. For `CL`, we used the same dedicated node on HoreKa as for the FL clients, utilizing one of the four available GPUs.

The energy consumption on the server for each algorithm was very low (14–29 kJ). Figure 2 represents the mean execution time and energy consumption of various FL algorithms for two clients. As shown in the figure, `Scaffold` demonstrates the highest execution time, resulting in the largest energy consumption across all sites. On average, it requires 103.33 min with a standard deviation of 8.84 min, highlighting its computational intensity. In our application, the energy consumption and runtime of `CL` compared to `FedAvg` show significant reductions. For energy consumption, `CL` consumes 382.279 kJ, whereas `FedAvg` consumes 1008.9152 kJ, resulting in a percentage reduction of approximately 163.97%. Similarly, for runtime, `CL` takes 643.245 s compared to 1687.0624 s for `FedAvg` leading to a percentage reduction of approximately 162.3%.

Reducing the available data per client dividing it by five leads to slower convergence. With five clients instead of two, each client contributes smaller updates per round, requiring more communication to reach the same performance. Since each client only accesses $\frac{1}{5}$ of the full data, we ensure a fair comparison with `CL` by maintaining the ratio $\frac{\texttt{num_rounds} \times \texttt{num_local_epochs}}{\texttt{num_clients}} = \texttt{num_epochs_CL}$. For these reasons, we first reduced the number of local epochs to five and increased the number of rounds to 13. In another experiment, we kept the local epochs at nine and increased the rounds to seven. Both setups achieved similar performance,

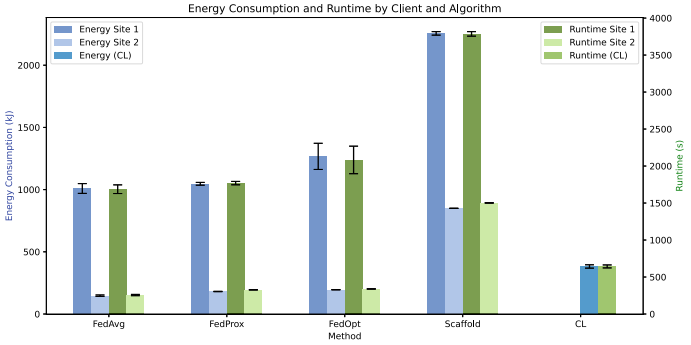


Fig. 2. Energy consumption and runtime by each site and each algorithm.

but with fewer epochs and more rounds, communication increased, leading to longer training time. Therefore, we settled on nine local epochs and seven rounds as a balanced choice. Table 4 shows the performance of the trained global model on the test dataset (KA+MU). When the dataset size is equal among clients, `FedProx` performed similar to `CL`. In `FedProx`, the aggregation on the server is done similarly to `FedAvg`. In the case of two clients, the global weights lean toward the MU dataset because of its larger weight. This means that when we compute the proximal term in `FedProx` the influence of the MU dataset is disproportionately high, potentially causing the corresponding client to pull the global model further away from the parameters of KA dataset. In contrast, in the *5-client scenario*, the MU dataset is divided among several clients. Consequently, the global weights become more reflective of the average of all local weights. This balanced aggregation results in a smaller difference between local and global models, thereby reducing the magnitude of the proximal term. With a lower proximal penalty, local updates are less constrained, which can lead to more effective convergence and improved overall performance in `FedProx` compared to the *2-client scenario*.

In this case, the performance of `Scaffold` dropped by 3–5% on each metric compared to `CL`. This decline may be due to the reduced dataset size available to each client, which adversely affects the estimation of the control variate.

`FedAvg`, achieving a mean weighted precision of 0.919 ± 0.012, while `FedOpt` still performed the worst among other FL algorithms. In `FedAvg`, the MU dataset features richer annotations and includes every class. In the *2-client scenario*, we assign greater weight to the MU dataset during aggregation, while the KA dataset receives less weight. However, in a *5-client scenario*, the MU dataset is split across several clients, resulting in it having the same weight as the KA dataset during aggregation. This difference in weighting may be a key factor contributing to the observed performance drop with five clients compared to two.

Table 4. Comparison of different FL algorithms and CL (overall) for five clients.

Metric	Centralized	FedAvg	FedProx	FedOpt	Scaffold
mwP	**0.939 ± 0.002**	0.919 ± 0.012	0.932 ± 0.003	0.891 ± 0.006	0.900 ± 0.002
mACC	**0.931 ± 0.002**	0.916 ± 0.013	0.926 ± 0.006	0.877 ± 0.006	0.879 ± 0.004
mwF$_1$	**0.928 ± 0.002**	0.904 ± 0.016	0.918 ± 0.008	0.865 ± 0.008	0.874 ± 0.002
mwIoU	**0.884 ± 0.003**	0.855 ± 0.019	0.872 ± 0.009	0.801 ± 0.009	0.810 ± 0.003

4.2 Comparison of the Workflow

In NVFlare, `CWT` workflow allows for configuring the order of clients during training. Specifically, the `cyclic_order` parameter can be set to determine whether the sequence of clients is fixed or random for each round. To investigate how the order of clients affects the performance of the model, especially with two clients where the dataset sizes differ, we conducted several experiments using both fixed and random orders.

Table 5 presents the overall results of training the model with two and five clients across different workflows. In *2-client scenario*, the `SL` workflow achieved similar performance to S&G (`FedAvg`), as expected: The only difference between these workflows is that in S&G, aggregation is performed on the server, whereas in `SL`, this task is done by one of the clients. In terms of execution time, the `SL` algorithm achieves a mean execution time of 949.86 s (for 2 clients), whereas `FedAvg` records a significantly higher mean execution time of 2906.07 s, representing an approximate 205.95% increase in training time. This is due to the elimination of communication overhead between clients and the server, which reduces latency and speeds up the training process in the `SL` workflow.

In `CWT`, we first train the model on the MU dataset as Client-1 and then transfer the weights of the model to Client-2 (KA dataset). In the next experiment, we also perform training in the reverse order, first on Client-1 (KA dataset) and then on Client-2 (MU dataset). The results of these experiments are presented in Table 5: We observe that training the model first on the KA dataset followed by MU (`CWT KA-MU`) consistently yields better performance across all metrics compared to training in the reverse order (`CWT MU-KA`). This suggests that KA with less data provides a stronger initial model foundation, leading to better generalization when fine-tuned on MU. Conversely, training on the larger MU dataset first, then on KA, can cause the model to overfit the smaller KA dataset.

In the case of five clients, we used a random order, and the performance was similar to that of the model trained with a fixed order (Client-1 to Client-5). Therefore, in `CWT`, the order of the clients should be carefully considered when the dataset sizes differ, as it can significantly impact the final model performance.

Furthermore, comparing different workflows, `SL` outperforming all the `CWT` and `DCWT` variations. This indicates that local aggregation in `SL` may provide better feature learning compared to `CWT`.

As shown in Table 5, for the *5-client scenario*, all frameworks exhibit statistically consistent performance across key metrics. Considering error ranges, the

results indicate no significant differences among the methods, suggesting that each approach is capable of delivering comparable results despite variations in their training and aggregation processes.

Table 5. Comparison of Different Workflows in FL.

	Two Clients						Five Clients		
Metric	S&G (FedAvg)	Swarm Learning	CWT (MU-KA)	CWT (KA-MU)	DCWT (MU-KA)	DCWT (KA-MU)	S&G (FedAvg)	Swarm Learning	CWT Random
mwP	0.939 ± 0.002	**0.949 ± 0.004**	0.926 ± 0.008	0.940 ± 0.002	0.920 ± 0.005	0.940 ± 0.009	0.919 ± 0.012	**0.939 ± 0.003**	0.935 ± 0.005
mACC	0.931 ± 0.002	**0.950 ± 0.004**	0.851 ± 0.031	0.901 ± 0.022	0.864 ± 0.011	0.920 ± 0.007	0.916 ± 0.013	**0.934 ± 0.003**	0.926 ± 0.004
mF_1	0.928 ± 0.002	**0.943 ± 0.005**	0.873 ± 0.022	0.926 ± 0.002	0.876 ± 0.006	0.921 ± 0.005	0.904 ± 0.016	**0.923 ± 0.005**	0.919 ± 0.005
mwIoU	0.884 ± 0.003	**0.910 ± 0.007**	0.796 ± 0.036	0.878 ± 0.003	0.802 ± 0.010	0.872 ± 0.008	0.855 ± 0.019	**0.883 ± 0.005**	0.875 ± 0.006

Figure 3 represents the mean execution time and energy consumption of various FL workflows for two sites. `SL` records a mean time execution of 952.42 s whereas `DCWT` records a significantly higher mean execution time of 1949.69 s. This represents an approximate 105% increase in training time. In `DCWT`, clients train the model sequentially, requiring each client to wait for the previous one to complete its training. As the number of clients increases, this sequential process leads to longer overall training times. Again, comparing `DCWT` to `CWT`, when communication time with the server is removed, `DCWT` demonstrates improved efficiency.

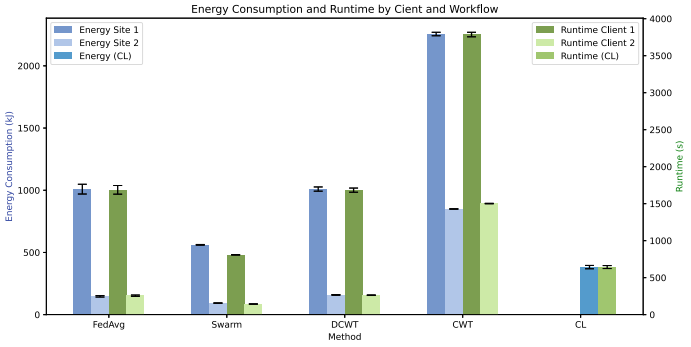


Fig. 3. Energy consumption and runtime by each site and each workflow.

Conclusions

In this work, we successfully transformed the introduced application from a traditional ML workflow into a FL workflow. The results show that FL algorithms can achieve comparable performance metrics to CL in detecting small objects within UAV images. `FedProx` shows variability in performance depending on client configurations. It performs poorly in the *2-client scenario* but substantially improves and achieves comparable results to CL in the *5-client scenario* due to reduced proximal penalties when data is more evenly distributed. `FedOpt` introduces additional hyperparameters, making it more challenging to set them optimally without prior data insights. `Scaffold`, on the other hand, can enhance performance but requires increased execution time and energy consumption, and in some cases, the improvements may not be significant enough to justify the additional cost. When the number of clients increases, convergence slows due to less data per client. Adjusting the training strategy by balancing the number of local epochs and aggregation rounds helps achieve optimal trade-offs between performance and computational cost.

Despite these promising findings, some limitations remain. Due to limited access to real-world data, scalability was tested only with a simulated setup of five clients, which may not capture real-world complexity. Moreover, while our study focuses on evaluating performance and system-level efficiency across various FL workflows, model interpretability was not within scope. In future work this aspect should be considered.

We also observed practical system-level challenges. Although HPC systems offer additional computational resources, batch job scheduling limits client availability, causing some clients to lack resources when others are ready.

In terms of sustainability, in our case study, when we compared to centralized training, experimental evidence shows FL can consume more energy than its centralized version in certain scenarios. Introducing decentralized workflows like `SL` and `DCWT` show a gain in shorter training time. Among FL workflows, `SL` demonstrates superior computational efficiency and runtime performance compared to S&G and `CWT` approaches. However, this relationship is not universal. It is highly depending on the setup, including the deployed machines, used algorithm/workflow and even the geographical region of the participating devices.

Acknowledgment. This work is supported by the AI4EOSC project, which receives funding from the European Union's Horizon Europe 2022 research and innovation programme under agreement 101058593. Additionally, computational resources were provided by the HoreKa supercomputer, funded by the Ministry of Science, Research and the Arts Baden-Württemberg and the Federal Ministry of Education and Research, and by the Helmholtz Association Initiative and Networking Fund through the Helmholtz AI platform grant and the HAICORE@KIT partition.

References

1. Agripina, N., Shen, H., Mafukidze, B.: Advances, challenges & recent developments in federated learning. Open Access Library J. **11**, 1–1 (2024)
2. AI4EOSC. https://ai4eosc.eu/
3. Amjoud, A.B., Amrouch, M.: Object detection using deep learning, CNNs and vision transformers: a review. IEEE Access **11**, 35479–35516 (2023). https://doi.org/10.1109/ACCESS.2023.3266093
4. Awasthi, R., et al.: Artificial intelligence in healthcare: 2023 year in review. medRxiv (2024). https://doi.org/10.1101/2024.02.28.24303482
5. Banabilah, S., Aloqaily, M., Alsayed, E., Malik, N., Jararweh, Y.: Federated learning review: fundamentals, enabling technologies, and future applications. Inf. Process. Manage. **59**(6), 103061 (2022)
6. Bjorck, N., Gomes, C.P., Selman, B., Weinberger, K.Q.: Understanding batch normalization. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31. Curran Associates, Inc. (2018)
7. bwCloud main page. https://www.bw-cloud.org/de/
8. Chang, K., et al.: Distributed deep learning networks among institutions for medical imaging. J. Am. Med. Inform. Assoc. **25**(8), 945–954 (2018)
9. Chellapandi, V.P., Yuan, L., Brinton, C.G., Żak, S.H., Wang, Z.: Federated learning for connected and automated vehicles: a survey of existing approaches and challenges. IEEE Trans. Intell. Veh. **9**(1), 119–137 (2024). https://doi.org/10.1109/TIV.2023.3332675
10. Chhikara, P., Tekchandani, R., Kumar, N., Tanwar, S.: Federated learning-based aerial image segmentation for collision-free movement and landing. In: Proceedings of the 4th ACM MobiCom Workshop on Drone Assisted Wireless Communications for 5G and Beyond, pp. 13–18. DroneCom 2021, Association for Computing Machinery, New York, NY, USA (2021)
11. Farsi, A.A., Khan, A., Rizwan, M., Bait-Suwailam, M.M.: Privacy and security challenges in federated learning for UAV systems: a comprehensive review (2024). https://doi.org/10.22541/au.172450870.03139596/v1
12. Feng, C., et al.: Assessing the sustainability and trustworthiness of federated learning models. Lynn and Bovet, Gérôme and Stiller, Burkhard, Assessing the Sustainability and Trustworthiness of Federated Learning Models (2025)
13. Grigorescu, S., Trasnea, B., Cocias, T., Macesanu, G.: A survey of deep learning techniques for autonomous driving. J. Field Robot. **37**(3), 362–386 (2020)
14. Guan, H., Yap, P.T., Bozoki, A., Liu, M.: Federated learning for medical image analysis: a survey. Pattern Recogn. **151**, 110424 (2024)
15. Gutiérrez Hermosillo Muriedas, J.P., Flügel, K., Debus, C., Obermaier, H., Streit, A., Götz, M.: perun: benchmarking energy consumption of high-performance computing applications. In: Cano, J., Dikaiakos, M.D., Papadopoulos, G.A., Pericàs, M., Sakellariou, R. (eds.) Euro-Par 2023: Parallel Processing, pp. 17–31. Springer Nature Switzerland, Cham (2023)
16. Haicore main page https://www2.helmholtz.ai/themenmenue/you-helmholtz-ai/computing-resources/index.html
17. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)
18. Roth, H.R., Cheng, Y., Wen, Y., Yang, I., et al.: NVIDIA FLARE: federated Learning from Simulation to Real-World (2023). https://doi.org/10.48550/arXiv.2210.13291

19. Horeka main page. https://www.scc.kit.edu/dienste/horeka.php
20. Kairouz, P., McMahan, H.B., et al.: Advances and open problems in federated learning. Found. Trends Mach. Learn. **14**, 1–210 (2021). https://doi.org/10.1561/2200000083
21. Kamp, M., Fischer, J., Vreeken, J.: Federated learning from small datasets (2023). https://arxiv.org/abs/2110.03469
22. Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S.J., Stich, S.U., Suresh, A.T.: Scaffold: stochastic controlled averaging for federated learning (2021). https://arxiv.org/abs/1910.06378
23. Li, Q., He, B., Song, D.: Model-contrastive federated learning . In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10708–10717. IEEE Computer Society, Los Alamitos, CA, USA (2021)
24. Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks (2020). https://arxiv.org/abs/1812.06127
25. Li, X., Jiang, M., Zhang, X., Kamp, M., Dou, Q.: FedBN: federated learning on non-IID features via local batch normalization. In: International Conference on Learning Representations (2021) https://openreview.net/forum?id=6YEQUn0QICG
26. Lu, Z., Pan, H., Dai, Y., Si, X., Zhang, Y.: Federated learning with Non-IID data: a survey. IEEE Internet Things J. **11**(11), 19188–19209 (2024)
27. Manthe, M., Duffner, S., Lartizien, C.: Federated brain tumor segmentation: an extensive benchmark. Med. Image Anal. **97**, 103270 (2024)
28. Mayer, Z., Epperlein, A., Vollmer, E., Volk, R., Schultmann, F.: Investigating the quality of UAV-based images for the thermographic analysis of buildings. Remote Sens. **15**(2), 301 (2023)
29. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.: Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR (2017). https://proceedings.mlr.press/v54/mcmahan17a.html
30. Miao, J., Yang, Z., Fan, L., Yang, Y.: FedSeg: class-heterogeneous federated learning for semantic segmentation. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8042–8052 (2023). https://doi.org/10.1109/CVPR52729.2023.00777
31. Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., Terzopoulos, D.: Image segmentation using deep learning: a survey. IEEE Trans. Pattern Anal. Mach. Intell. **44**(7), 3523–3542 (2022)
32. Mlflow. https://mlflow.org/
33. European Parliament - European Union: Regulation (EU) 2016/679 of the european parliament and of the council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (general data protection regulation). https://eur-lex.europa.eu/eli/reg/2016/679/oj (2016). Accessed 17 Jan 2025
34. Qiu, X., et al.: A first look into the carbon footprint of federated learning. J. Mach. Learn. Res. **24**(1), 1–23 (2024)
35. Rashidi, G., Bounias, D., Bujotzek, M., Mora, A.M., Neher, P., Maier-Hein, K.H.: The potential of federated learning for self-configuring medical object detection in heterogeneous data distributions. Sci. Rep. **14**(1), 23844 (2024)
36. Reddi, S.J., et al.: Adaptive federated optimization. In: International Conference on Learning Representations (2021). https://openreview.net/forum?id=LkFG3lB13U5

37. Ronneberger, O., Fischer, P., Brox, T.: U-net: convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, pp. 234–241. Springer, Cham (2015)
38. Slurm workload manager documentation. https://slurm.schedmd.com/documentation.html
39. Vollmer, E., et al.: Enhancing UAS-based multispectral semantic segmentation through feature engineering. IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens. **18**, 6206–6216 (2025)
40. Vollmer, E., Klug, L., Volk, R., Schultmann, F.: AI in multispectral image analysis: implementing a deep learning model for the segmentation of common thermal urban features to assist in the automation of infrastructure-related maintenance. Presentation at the 4th AI in AEC Conference, Helsinki, Finland (2024). https://doi.org/10.5445/IR/1000169834
41. Vollmer, E., et al.: Thermal urban feature segmentation - Multispectral (RGB + thermal) UAS-based images from Germany with annotations (2025). https://doi.org/10.5281/zenodo.10814413
42. Vollmer, E., Ruck, J., Volk, R., Schultmann, F.: Detecting district heating leaks in thermal imagery: comparison of anomaly detection methods. Autom. Constr. **168**, 105709 (2024)
43. Vollmer, E., Volk, R., Schultmann, F.: Automatic analysis of UAS-based thermal images to detect leakages in district heating systems. Int. J. Remote Sens. **44**, 31 (2023)
44. Wang, F., Gursoy, M.C., Velipasalar, S.: Feature-based federated transfer learning: communication efficiency, robustness and privacy. IEEE Trans. Mach. Learn. Commun. Network. **2**, 1 (2024)
45. Warnat-Herresthal, S., Schultze, H., Shastry, K.: Swarm learning for decentralized and confidential clinical machine learning. Nature **594**(7862), 265–270 (2021)
46. Wu, P., Zhang, Z., Peng, X., et al.: Deep learning solutions for smart city challenges in urban development. Sci. Rep. **14**(1), 5176 (2024)
47. Wu, Y., He, K.: Group normalization (2018). https://arxiv.org/abs/1803.08494