FedEdge: Federated Learning with Docker and Kubernetes for Scalable and Efficient Edge Computing

Mir Hassan Leonardo Lucio Custode Kasim Sinan Yildirim Giovanni Iacca

Department of Information Engineering and Computer Science, University of Trento, Italy
{mir.hassan, leonardo.custode, kasimsinan.yildirim, giovanni.iacca}@unitn.it

Abstract

This paper introduces a novel framework that integrates Docker and Kubernetes to address well-known challenges in federated learning. Federated learning has gained significant attention as a privacy-preserving and scalable machine learning paradigm. However, existing frameworks often lack portability, scalability, resource efficiency, fault tolerance, standardization, and ecosystem integration. To overcome these limitations, we propose a conceptual framework that combines Tensorflow FL with Docker's containerization capabilities and Kubernetes' orchestration capabilities. Our approach fills all the gaps in existing FL frameworks. By leveraging Docker containers, our model achieves efficient resource allocation, maximizing computing resources while maintaining portability and scalability. Kubernetes further enhances resource allocation by orchestrating the deployment of these containers, minimizing resource consumption. Our proposed framework provides opportunities for large-scale distributed machine learning applications, enabling the widespread use of federated learning methodologies.

1 Introduction

Internet of Things (IoT) devices are becoming ubiquitous in our everyday lives. Moreover, thanks to the recent developments in Machine Learning (ML), these devices can aid humans in a wide variety of tasks. However, since IoT devices use information from different sensors that may potentially perceive sensitive data, they pose a risk for the privacy of the users. Federated learning (FL) bears the promise of addressing this issue by enabling the training of ML models on edge devices [1]. More specifically, FL is a distributed approach to ML models that allows the training of ML models on local devices without transmitting the raw data to a central server. In fact, the models' parameters are updated by aggregating information about the parameters of the individual models on the edge, rather than their data. The local edge devices training process is repeated until the global model converges to an optimal solution [2]. In FL, clients' sensitive data are thus effectively preserved, as no private data is shared among the edge devices and the central server.

However, existing federated learning frameworks face portability, resource allocation, fault tolerance, and standardization challenges, limiting their scalability and effectiveness. Existing frameworks such as PySyft, Flower, Paddle FL, and Intel OpenFL suffer from a lack of portability and fault tolerance and require additional integration support. The problem addressed in this research is the need for a comprehensive and efficient framework that overcomes these challenges and provides a robust solution for federated learning. Existing FL frameworks lack mechanisms to allocate computing resources optimally, handle device failures, and ensure consistent implementation across various scenarios. These limitations hinder the widespread adoption of federated learning in real-world applications.

Deploying such a framework for FL systems requires mechanisms to tackle the following issues:

1. **Portability of the framework to heterogeneous devices**: In a heterogeneous system, the specific ML model should be adapted to the capabilities of each device, e.g. by using quantized or pruned versions of the global model;
2. **Portability**: In distributed systems, maneuverability and compatibility is enabled in applications of different heterogeneous systems.
3. **Fault tolerance**: If a fault happens in one of the edge nodes, the infrastructure should be able to repair it;
4. **Model standardization and versioning**: All the nodes should run different versions of the same model, which requires central coordination to aggregate the information coming from each of the edge nodes, and eventually deploy the resulting model;
5. **Scalability**: The resource allocation mechanism must scale effectively as the number of edge devices and workload complexity increases. This involves designing an allocation algorithm that can handle large-scale deployments and accommodate the growing demands of the system without sacrificing performance and efficiency.

Our research aims to address these limitations and provide an advanced framework for federated learning by integrating Docker and Kubernetes. We aim to enhance resource allocation, fault tolerance, and portability in federated learning. Docker provides a lightweight and portable containerization solution, facilitating the deployment of machine learning models on diverse edge devices. Kubernetes offers powerful container orchestration capabilities, enabling efficient utilization of computing resources. The use of Kubernetes and Docker can in fact simplify the deployment and management of FL models on the edge devices while providing a scalable and secure platform for FL [3]. We should note that some approaches leveraging Kubernetes and Docker for FL have been already proposed in the recent literature. A useful federated learning framework is created in [4, 5] to facilitate the deployment, aggregation, and device monitoring of models. According to [6], a federated learning method that focuses on the on-demand client delivers a greater amount and heterogeneity of data for the learning process. Due to its evolutionary technique, the Genetic Algorithm (GA) is used in [7] to solve the multi-objective optimization problem that represents the deployment of the client and model.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 describes the proposed architecture and framework to be used in the implementation of FedEdge. Section 4 presents the applications for the proposed framework. In Section 5, we have concluded and described the future work.

2 Related work

Edge Computing (EC) is a paradigm that allows computing tasks to be performed close to the data source, instead of transmitting data to data centers or the cloud. By processing data locally, EC enables edge nodes to respond to service requests quickly, thus reducing bandwidth usage and network latency [8]. This approach is especially relevant in the context of the IoT, where a vast number of interconnected devices are used by people on a daily basis [9]. EC can dramatically improve location and context-aware apps as it is placed closer to user devices and data sources. It has been acknowledged as a key technology in 5G networks, supporting a number of network applications [10], and it is projected to be a key element in the upcoming 6G network. However, the open-source EC ecosystem is currently experiencing its most competitive growth phase, with numerous open-source platforms, portability measures, and convergence initiatives. This trend underscores the pressing need to develop more efficient systems that can handle the surging demand for cloud computing on the edge [8].

Recently, container technology [11] has gained significant attention in the domain of cloud services. Previously, server virtualization technology [12] had been extensively used in this field. Similarly to server virtualization, container technology also provides the benefit of resource isolation and allocation. Containers package code, system libraries, and configurations required in a lightweight, standalone executable package. Typically, a container instance is only a few dozen MB in size. This allows the creation of numerous servers from a single server. In contrast, virtualization is an abstraction of physical hardware that frequently takes up several GB of space and includes a full copy of the operating system [12, 13].

Kubernetes, often abbreviated as K8s, is a cluster management system that is available as an open-source tool. This system can automatically deploy, extend, and maintain a cluster of containers. It was developed by Google in 2014 as an open-source version of the Borge system [14]. The Kubernetes architecture comprises Master components and Node components. The hosts that have the master components installed are referred to as the Masters, while the hosts with node components are labeled as Nodes. The Master is the central computer controlling the entire platform's operation. Kubectl, API-Server, Etcd, Controller Manager, Scheduler, and add-ons are among its components. API-Server provides a variety of interfaces for executing resource operations, whereas Kubectl is a management tool for managing Kubernetes cluster data. The scheduler oversees the scheduling of resources, and the controller manager ensures that the facilities and pod details (a "pod" is a fundamental unit that provides a service) are in real-time linked by implementing multiple controls [3]. Additionally, the plug-in Addons provide a user interface for monitoring information on the Kubernetes platform. Node components collaborate with the Master to manage the server. These components include kubelet, which manages the Node's pod operation and upkeep. Kub-proxy manages communication within the cluster between the pods, while Cluster denotes all the pods linked to the same Master.

2.1 Federated Learning

FL depends on synchronized computing (for client model training and server model aggregation) and communication (for local model uploads and aggregated model distribution to clients). Thus, communication and computational efficiency are crucial for FL within an EC context [15, 16]. Figure 1 presents a typical FL scheme consisting of clients (data owners) engaging in cooperative learning and an FL server (model owner) overseeing the process. The FL process can be seen as a sequence the following phases:

Step 1 The server sends an initial model to the clients.

Step 2 Each client refines its own model, based on its data.

Step 3 Each client sends a revised version of the model to the server.

Step 4 The server combines all the versions in a new model.

Step 5 Restart from Step 1.

A comparison of the existing open-source frameworks for FL supporting Docker and Kubernetes is shown in Table 1. In the following, we summarize the main aspects of each of these frameworks. In the table, we compare the existing frameworks w.r.t. the following properties:

- PyTorch support: PyTorch [17] is one of the most popular Deep Learning frameworks. Thus, supporting this library is crucial for compatibility with a wide variety of deep learning algorithms;
- Tensorflow (TF) and Keras support: TF [18] and Keras [1] are other widely used deep learning libraries;

[1]https://keras.io/

Table 1. Open-source frameworks based on Kubernetes and Docker for Federated Learning

Framework	ML libraries			Portability	Fault tolerance	Standardization	Ecosystem Integration	Maintained	Batching
	PyTorch	TF + Keras	Scikit-Learn						
Tensorflow FL	✗	✓	✓	✗	✓	✓	✗	✓	✓
PySyft	✓	✓	✗	✗	✗	✗	✓	✗	✓
Flower	✓	✓	✓	✓	✓	✓	✗	✓	✗
PaddleFL	✗	✗	✗	✓	✓	✓	✓	✗	✓
Intel OpenFL	✓	✓	✓	✗	✓	✗	✓	✓	✗
FedEdge (Ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓

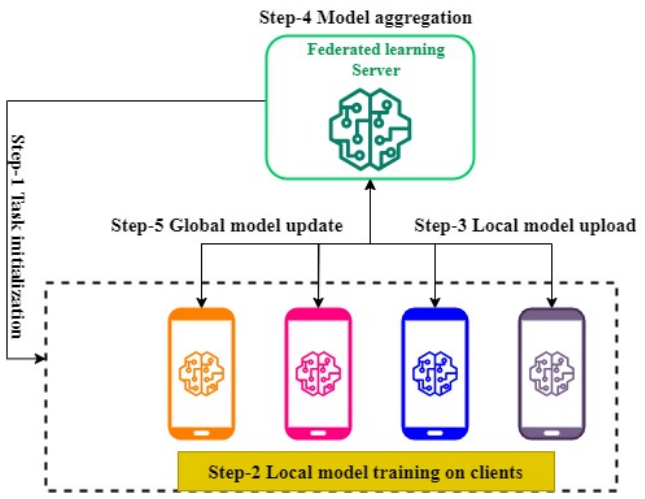
Step-4 Model aggregation
Federated learning Server
Step-1 Task initialization
Step-5 Global model update
Step-3 Local model upload
Step-2 Local model training on clients

Figure 1. Conceptual scheme of Federated Learning.

- Scikit-Learn support: Scikit-Learn [19] is one of the most popular general machine learning libraries. Therefore, it is necessary to support this library to equip our framework with a wide variety of well-tested ML algorithms;
- Portability to heterogenous devices: Our framework should be able to tailor ML models to the capabilities of each edge device;
- Fault tolerance: The framework must be able to repair faults occurring at the edge;
- Portability: It is crucial that all the nodes in the network use the same model, even if using different (tailored) versions;
- Scalability: Scaling is a fundamental property in smart distributed systems, as it ensures that the system can keep working even when the number of devices grows;
- Maintainance: Too ensure compatibility, the library should be maintained;
- Ecosystem integration:Docker and Kubernetes have a vast ecosystem of tools and services that can be integrated with TensorFlow Federated, such as logging, monitoring, and security solutions, enhancing the overall functionality and manageability of the federated learning system.

2.1.1 TensorFlow FL

TensorFlow[2] is a widely used framework that integrates seamlessly with the TensorFlow ecosystem. It offers portability, fault tolerance, and standardization support. However, it has limited resource efficiency and orchestration in distributed and scalable environments.

2.1.2 PySyft

PySyft[3] is a powerful framework built on top of PyTorch, enabling FL and securing multi-party computation. It provides privacy-preserving techniques and supports decentralized training, but it lacks standardization and may require additional effort for integration.

2.1.3 Flower

Flower[4] is an open-source framework that simplifies FL system development. It offers flexibility in ML library choices and provides fault tolerance mechanisms. However, it has limited standardization and may require intervention for resource allocation.

2.1.4 PaddleFL

PaddleFL[5] is an FL framework developed by PaddlePaddle. It supports efficient large-scale FL and provides fault tolerance mechanisms. However, it has limited standardization and may require customization for specific use cases.

2.1.5 Intel OpenFL

Intel OpenFL [6] is an FL framework for privacy-preserving collaborative AI. It offers privacy-enhancing techniques and supports distributed learning. However, it may require additional effort for integration and lacks comprehensive standardization.

3 Proposed FedEdge framework and tools

As discussed earlier, this research aims to design a framework that enhances the edge network efficiency and guarantees proper resource allocation using Docker and Kubernetes for deployments of FL schemes. Therefore, the proposed FedEdge framework consists of two main components:

- **Edge Devices**: These devices host data that is used to train the ML models. Each device will be equipped with Kubernetes and Docker, enabling the deployment of containers for efficient resource management.
- **Central Server**: The central server store the global model and provide an initial model for edge devices to train on.

[2]https://www.tensorflow.org/federated
[3]https://github.com/OpenMined/PySyft
[4]https://flower.dev/
[5]https://github.com/PaddlePaddle/PaddleFL
[6]https://github.com/securefederatedai/openfl

It also manage the distribution of the global model to edge devices for further training.

The use of Kubernetes (or more lightweights distributions such as K3s[7]) and Docker in the proposed framework enables easy deployment, scalability, and portability. The proposed solution can be easily evaluated using real-world data, allowing us to compare the performance of a FL approach (see Figure 2) to that of a traditional ML approach.

The proposed approach is graphically described in Figure 3, which illustrates the integration of Kubernetes and Docker within the FedEdge framework.

3.1 Implementation

In this section, we explain the framework and tools that can be used to implement the proposed FedEdge infrastructure.

3.1.1 Framework

3.1.1.1 TensorFlow FL

TensorFlow is a strong contender among the existing FL frameworks. It offers seamless integration with popular ML libraries like TensorFlow and Keras, ensuring compatibility and flexibility in model deployment. In our framework, TensorFlow FL is the main building block, which allows us to perform FL. Integrating Docker and Kubernetes with TFF enhances portability, scalability, resource efficiency, fault tolerance, and ecosystem integration, making it a powerful combination for deploying and managing large-scale federated learning applications.

- ***Portability:*** Docker enables easy deployment of TensorFlow Federated (and ML models) across different environments, while Kubernetes allows for seamless management of the application.
- ***Scalability:*** Kubernetes enables easy scaling of TensorFlow Federated to handle large-scale federated learning tasks.
- ***Fault tolerance:*** Kubernetes provides fault tolerance features, ensuring the continuous availability of TensorFlow Federated.
- ***Ecosystem integration:*** Docker and Kubernetes have a vibrant ecosystem, offering various tools and services that can be leveraged for monitoring, logging, security, and more.

3.1.2 Tools

3.1.2.1 Kubeflow

To create and deploy ML workflows, including FL workflows, Kubeflow[8] is an open-source ML toolbox for Kubernetes. Kubeflow offers a scalable and flexible solution to deploy and maintain ML models on edge devices while maximizing communication and compute efficiency.

3.1.2.2 Seldon

FL on the edge can be implemented using Seldon[9], an open-source framework for deploying and managing ML models on Kubernetes. Seldon provides various features that can be used to enhance ML workflows on edge devices, including model versioning, monitoring, and scaling.

[7]https://k3s.io

[8]https://www.kubeflow.org

[9]https://www.seldon.io

3.1.2.3 Istio

Microservices on Kubernetes can be managed and secured using the open-source service mesh technology Istio[10]. Istio can offer a method for streamlining communication across various FL process components on edge devices while preserving data confidentiality and privacy.

3.1.2.4 Harbor

For organizing and storing container images used in FL workflows on Kubernetes, Harbor[11] is an open-source container registry. Harbor provides various features that can be used to ensure the consistency and security of container images on edge devices, including role-based access control and vulnerability testing.

3.1.2.5 KubeEdge

The open-source EC platform KubeEdge can be used to execute containerized workloads on edge devices, including FL workloads. KubeEdge[12] provides various capabilities that can be used to enhance communication and computation on edge devices, including edge intelligence and device management.

4 Applications

In this section, we describe a number of potential applications for the proposed infrastructure.

4.1 Smart buildings

Edge devices in a smart building, such as temperature, motion, and occupancy sensors, can collect a lot of data that can be utilized to increase energy efficiency and the well-being of people in an indoor environment. ML models can be trained locally on sensor data using FL at the edge, protecting user privacy and consuming less network bandwidth. Once the model is trained, it can be used to enhance building management systems e.g. for optimized lighting and HVAC (heating, ventilation, and air conditioning).

4.2 Traffic management

The condition of traffic, the state of the roads, and the behaviour of automobiles can all be observed using edge devices such as traffic cameras, GPS units, and car sensors in a smart transportation system. FL on the edge can make it possible to train ML models locally on the data, ensuring privacy while also consuming less network bandwidth. The trained models can then be used to improve the timing of traffic lights, reroute cars, and find and predict crashes and traffic jams.

4.3 Health monitoring

The health and behaviour of patients can be studied using edge devices in healthcare, such as wearable, smart medical equipment, and medical sensors. FL on the edge can enable ML models to be trained locally on the data, ensuring privacy while also consuming less network bandwidth. The trained models can then be used to find and predict health problems, check if patients take their medications, and give specific health advice.

[10]https://istio.io

[11]https://goharbor.io

[12]https://kubeedge.io/en/

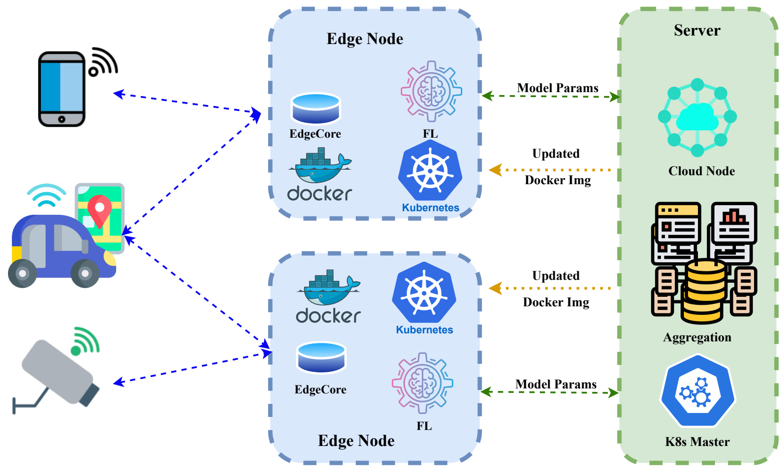

Figure 2. Conceptual scheme of the proposed FedEdge infrastructure.

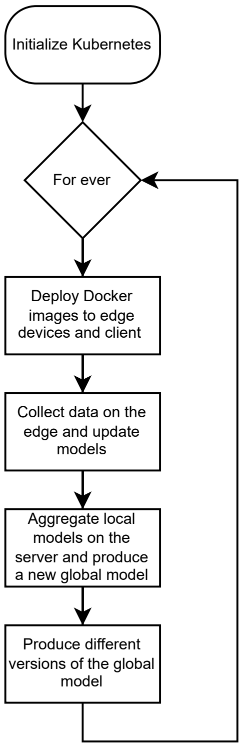
Initialize Kubernetes
For ever
Deploy Docker images to edge devices and client
Collect data on the edge and update models
Aggregate local models on the server and produce a new global model
Produce different versions of the global model

Figure 3. Flow chart of the proposed method.

4.4 Industrial Internet of things

In a smart industrial system, edge devices such as sensors, cameras, and robots can collect data on workers' behavior, equipment condition, and production processes. FL at the edge can enable ML models to be trained locally on the data, ensuring privacy and reducing network traffic simultaneously. Once trained, the models can be used to increase production efficiency, identify and stop machine faults, and ensure worker safety.

5 Conclusions and future works

In this paper, we presented the FedEdge framework, a conceptual solution for implementing intelligent distributed systems, such as smart buildings, smart traffic management systems, smart healthcare monitoring systems, and intelligent transportation systems, through Federated Learning integration on large-scale applications. The FedEdge framework, which integrates Docker and Kubernetes with TensorFlow Federated, presents a compelling solution for addressing the challenges of portability, scalability, fault tolerance, and ecosystem integration in federated learning systems. By leveraging the containerization and orchestration capabilities provided by Docker and Kubernetes, we have demonstrated improved deployment flexibility, efficient resource allocation, seamless scalability, robust fault tolerance, and enhanced integration with existing tools and services. These advantages position our framework as a promising approach for researchers and practitioners in the field of federated learning. Future work will focus on further optimizing resource allocation strategies, enhancing fault tolerance mechanisms, and exploring novel techniques for performance optimization.

Acknowledgments

Funded by the European Union (project no. 101071179). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or EISMEA. Neither the European Union nor the granting authority can be held responsible for them.

6 References

[1] Federated Learning | TensorFlow Federated. `https://www.tensorflow.org/federated/federated_learning`.

[2] Yu, Rong and Li, Peichun. Toward resource-efficient federated learning in mobile edge computing. *IEEE Network*, 35(1):148–155, 2021.

[3] Nguyen, Quang-Minh and Phan, Linh-An and Kim, Taehong. Load-balancing of Kubernetes-based edge computing infrastructure using resource adaptive proxy. *Sensors*, 22(8):2869, 2022.

[4] Kim, Jingyeom and Kim, Doyeon and Lee, Joohyung. Design and Implementation of Kubernetes enabled Federated Learning Platform. In *International Conference on Information and Communication Technology Convergence*, pages 410–412, 2021.

[5] Park, Jongbin and Woo Kum, Seung. Design and Development of Server-Client Cooperation Framework for Federated Learning. In *International Conference on Ubiquitous and Future Networks*, pages 271–273, 2022.

[6] Chahoud, Mario and Otoum, Safa and Mourad, Azzam. On the feasibility of Federated Learning towards on-demand client deployment at the edge. *Information Processing & Management*, 60(1):103150, 2023.

[7] Mario Chahoud, Hani Sami, Azzam Mourad, Safa Otoum, Hadi Otrok, Jamal Bentahar, and Mohsen Guizani. On-demand-fl: A dynamic and efficient multi-criteria federated learning client deployment scheme. *IEEE Internet of Things Journal*, 2023.

[8] Karanika, Anna and Soula, Madalena and Anagnostopoulos, Christos and Kolomvatsos, Kostas and Stamoulis, George. Optimized analytics query allocation at the edge of the network. In *International Conference on Internet and Distributed Computing Systems*, pages 181–190. Springer, 2019.

[9] Bonomi, Flavio and Milito, Rodolfo and Natarajan, Preethi and Zhu, Jiang. Fog computing: A platform for Internet of Things and analytics. *Big data and Internet of Things: A roadmap for smart environments*, pages 169–186, 2014.

[10] Pham, Quoc-Viet and Fang, Fang and Ha, Vu Nguyen and Piran, Md Jalil and Le, Mai and Le, Long Bao and Hwang, Won-Joo and Ding, Zhiguo. A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art. *IEEE Access*, 8:116974–117017, 2020.

[11] Pahl, Claus and Lee, Brian. Containers and clusters for edge cloud architectures–a technology review. In *International Conference on Future Internet of Things and Cloud*, pages 379–386. IEEE, 2015.

[12] Barr, Ken and Bungale, Prashanth and Deasy, Stephen and Gyuris, Viktor and Hung, Perry and Newell, Craig and Tuch, Harvey and Zoppis, Bruno. The VMware mobile virtualization platform: is that a hypervisor in your pocket? *ACM SIGOPS Operating Systems Review*, 44(4):124–135, 2010.

[13] Liu, Shaoshan and Liu, Liangkai and Tang, Jie and Yu, Bo and Wang, Yifan and Shi, Weisong. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.

[14] Phan, Linh-An and Kim, Taehong and others. Traffic-aware horizontal pod autoscaler in Kubernetes-based edge computing infrastructure. *IEEE Access*, 10:18966–18977, 2022.

[15] Duan, Qiang and Huang, Jun and Hu, Shijing and Deng, Ruijun and Lu, Zhihui and Yu, Shui. Combining Federated Learning and Edge Computing toward Ubiquitous Intelligence: Challenges, Recent Advances, and Future Directions, 2022.

[16] Park, Jongbin and Kum, Seung Woo. Design and Development of Server-Client Cooperation Framework for Federated Learning. In *International Conference on Ubiquitous and Future Networks*, pages 271–273. IEEE, 2022.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[18] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.