Scaling machine learning at the edge-cloud: a distributed computing perspective

Fabrizio Marozzo, Alessio Orsino, Domenico Talia, Paolo Trunfio
DIMES
University of Calabria
Rende, Italy
{fmarozzo, aorsino, talia, trunfio}@dimes.unical.it

***Abstract*—The widespread diffusion of Internet of Things (IoT) devices has led to an exponential growth in the volume of data generated at the edge of the network. With the rapid spread of machine learning (ML)-based applications, performing compute and resource-intensive learning tasks at the edge has become a critical issue, resulting in the need for scalable and efficient solutions that can overcome the resource constraints of edge devices. This paper analyzes the problem of scaling ML applications and algorithms at the edge-cloud continuum from a distributed computing perspective. In particular, we first highlight the limitations of traditional distributed architectures (e.g., clusters, clouds, and HPC systems) when running ML applications that use data generated at the edge. Next, we discuss how to enable traditional ML algorithms combining the benefits of edge computing, such as low-latency processing and privacy preservation of personal user data, with those of cloud computing, such as virtually unlimited computational and storage capabilities. Our analysis provides insights into how properly separated parts of a ML application can be deployed across edge-cloud architectures in order to optimize its execution. Moreover, examples of ML applications and algorithms appropriately adapted for the edge-cloud continuum are shown.**

***Index Terms*—Machine learning, distributed machine learning, Internet of Things, edge computing, cloud computing, edge-cloud continuum**

I. Introduction

Artificial intelligence (AI), especially machine and deep learning, has become a widely adopted solution for various everyday tasks such as speech recognition, email spam filtering, and fraud prevention [1]. The availability of massive amounts of data, the so-called big data, has played a crucial role in the advancement of machine learning (ML) solutions. By leveraging large datasets and increased computational capabilities of modern hardware, machine learning can extract valuable insights from data to aid decision-making [2]. While conventional distributed systems have proven effective for performing complex learning tasks [3], they may not be suitable to meet the low-latency requirements of ML-based real-time applications. As a result, novel approaches such as edge computing, in which data is processed closer to the source, have emerged as promising solutions to address this challenge. In fact, the proliferation of Internet of Things (IoT) devices, such as smart cameras, wearables, and smartphones, has led to a significant increase in data generation at the network edge, which refers to the point of connection between a device and the Internet. Transmitting this data from edge devices to a centralized server for collection, processing, and analysis incurs high communication costs and can impact latency, which is critical for applications like health monitoring and autonomous vehicle steering.

However, IoT devices at the network edge face limitations in computational power, energy, storage capacity, and bandwidth. These devices are considered resource-constrained, making them impractical to run resource-intensive machine learning tasks. The resource-constraint nature of edge devices requires integration with cloud computing platforms, which allow for persistent aggregation of big data and compute-intensive analyses using large computing resources. This approach leads to the concept of the edge-cloud compute continuum. Consequently, recent efforts have focused on adapting machine learning algorithms to enable cooperative training between edge and cloud or among edge devices themselves [4]. This turns out to be quite a challenge due to the limited capabilities of edge devices, the different hardware and technologies that edge and cloud employ, and the absence of standardized software stacks for efficient management. Moreover, ensuring security and privacy becomes crucial when transferring data to other devices or remote servers for parallel model training. There are also important technical challenges to be addressed in communication, as the bandwidth between edge devices can be significantly slower than local computation time, necessitating the development of communication-efficient methods for the training process.

In this paper we analyze how distributed machine learning applications and algorithms can be efficiently executed at the edge-cloud continuum. In particular, a machine learning application can be seen as a flow of steps, such as data collection, compression, preparation, and local learning. Some of these steps that require low response times and real-time analytics can be performed on the edge, while steps that require large amounts of computational resources, access to large datasets, and data aggregation, need to be performed on the cloud. Machine learning applications can also be broken down to an even lower level, as a flow of data transformation tasks such as map, filter, and reduce tasks that can be performed on the data. In an edge-cloud continuum architecture, transformation tasks that perform local processing are better suited for the device or edge layers, while tasks that involve data shuffling across partitions are more suitable for the fog or cloud layers. To

show our approach applied to practical examples, the task flow diagrams of two machine learning algorithms (k-means and Random Forest) are described, and for each task is indicated which levels of an edge-cloud architecture are best suited to execute it.

The structure of the paper is as follows. Section II analyzes the related machine learning techniques and solutions in three specific domains of distributed computing: (i) distributed machine learning on High Performance Computing systems; (ii) machine learning on resource-constrained edge devices; and (iii) federated learning. Section III discusses how a machine learning application can be performed on an edge-cloud continuum architecture. Section IV describes how the tasks of two machine learning algorithms can be distributed on the different layers of an edge-cloud continuum architecture. Finally, Section V concludes the paper.

II. Related work

While previous studies have focused heavily on distributed deep learning models for edge environments, there is a need for further exploration and adaptation of traditional machine learning algorithms, which are commonly used in AI-based applications [5]. Solutions and technologies for deep neural network model training and inference, as well as learning in resource-constrained edge computing environments, have been investigated in different survey works [6], [7]. However, while deep learning was extensively covered, the other machine learning algorithms were only briefly mentioned. Imteaj et al. [8] explored the federated learning paradigm and the challenges of training distributed machine learning models for resource-constrained IoT devices. Rosendo et al. [9] analyzed machine learning, data analytics, and frameworks for big data processing in edge, cloud, and edge-cloud architectures. Since the goal of this paper is to describe how traditional machine learning algorithms can be effectively deployed on edge-cloud continuum architectures, below we describe the main machine learning techniques and solutions in three specific domains of distributed computing: (i) distributed machine learning on HPC; (ii) machine learning on resource-constrained edge devices; and (iii) federated learning.

a) Distributed machine learning on HPC: Machine learning algorithms are typically designed to run on high-performance machines, as training data and machine learning models grow in size, learning on a single machine is unsuitable [10]. This problem can be solved by using distributed computing, where multiple computing nodes collaborate to train a model in parallel. There are two ways to achieve this: distributing the data on worker nodes, which execute the same algorithm on different data partitions, or distributing the model, in such a way that nodes process the same data but execute different sections of the model, and local results are then aggregated. In both strategies, worker nodes can be organized in either a centralized architecture, where the learning process is performed iteratively by updating and synchronizing model parameters on a central server, or in a decentralized one, where each worker node communicates with others and the model is aggregated without involving a central node. In all approaches and architectures, distributed learning avoids the need to collect large volumes of data on a single machine [11]. Different approaches have been suggested to accelerate traditional machine learning algorithms on distributed HPC infrastructures using big data analysis frameworks like Apache Hadoop and Spark [2]. For instance, parallelized versions of SVM [12] and k-means [13] have been proposed using MapReduce, whereas Spark has been leveraged for parallelized versions of Random Forest [14] and DBSCAN [15]. Over the years, libraries have been developed with the main implementations of distributed machine learning algorithms. For instance, Apache Mahout [16] is an open-source library for developing scalable machine learning algorithms in Hadoop, including recommendation mining, clustering, classification, and frequent itemset mining. On the other hand, MLlib [17], Apache Spark's machine learning library, provides parallel machine learning algorithms such as classification, regression, clustering, and collaborative filtering. Although these libraries provide the main implementations of machine learning algorithms, they have been designed to run on uniform distributed computing environments such as those provided by clusters or HPC systems. To be efficiently exploited in heterogeneous distributed computing environments (i.e., computation/storage capacity, hardware, software stacks) such as that of the edge-cloud continuum, they must be appropriately adapted otherwise they are unusable.

b) Machine learning on resource-constrained edge devices: The deployment of machine learning applications at the edge presents a significant opportunity for several real-world scenarios, benefiting from the low latency associated with performing on-device training and inference close to the data sources. However, the limited computational, memory, and energy resources, hardware heterogeneity, and security and communication concerns of IoT devices pose a significant challenge to performing heavy learning tasks on them. Most techniques for edge training focus on deep learning, primarily using the gradient-descent technique. For example, Wang et al. [18] proposed a technique that trains deep learning models at the edge by performing local gradient descent on multiple edge devices and sending local models to an aggregator that computes a weighted average, which is sent back to all edge devices for the next iterations. Another approach is to train large and accurate models on high-performance machines and then use compression techniques, such as low-rank approximation, knowledge distillation, pruning, and parameter quantization, to reduce model size. However, smaller models often result in lower accuracy, thus the trade-off between accuracy and costs must be carefully considered. To address these challenges, the EdgeML [19] library provides a set of open-source algorithms for building machine learning models that can run directly on edge devices, with much lower memory requirements than traditional ML algorithms. The trained models, which include tree-based classifiers [20], k-nearest neighbors (kNN) classifiers [21], and recursive neural networks (RNNs) [22], can be loaded onto edge devices, such as IoT devices and sensors, to

make fast and accurate predictions. The TinyML project [23] and Tensorflow Lite Micro [24] are dedicated to optimizing deep learning inference for edge devices with limited memory, such as microcontrollers, enabling the efficient deployment of AI applications in the context of edge. However, these solutions are primarily designed to perform learning directly on the end devices with pruned models and do not allow the use of global and complex models distributed across devices, edges and clouds.

c) Federated learning: Traditional approaches to distributed machine learning often neglect privacy and security concerns during training and inference, relying on centralized data management. However, this becomes a problem when edge devices with limited defense capabilities are involved or sensitive data need to be sent to remote servers. To address this issue, federated learning has emerged as a paradigm for training centralized models while keeping data distributed across devices, allowing to never send data produced at the edge devices to a centralized node, thus preserving privacy and also reducing latency. For instance, Kumar et al. [25] proposed applying the Federated Averaging technique to the distributed k-means algorithm, ensuring privacy preservation and latency reduction. Other efforts have been devoted to applying the federated learning paradigm to ensemble techniques, especially Random Forest [26], [27], without the need for exchanging raw data. Among libraries implementing federated learning, FedML [28] is an open research library designed to facilitate the development of federated learning algorithms on top of three computing paradigms: on-device training for edge devices, distributed computing, and single-machine simulation. Although the solutions discussed here allow for the distribution of training and inference operations across edge and cloud levels, they cannot be applied to all classes of machine learning algorithms. Indeed, some algorithms need to be properly optimized to run in a combined and efficient way on all levels of edge-cloud continuum architectures (i.e., device, edge, fog, and cloud).

III. Distributed machine learning at the edge-cloud continuum

In this section we discuss how to effectively deploy machine learning applications across different layers of an edge-cloud continuum architecture. In particular, a machine learning application can be broken down with different levels of granularity. It can be seen either as a flow of *steps*, such as data collection, compression, preparation, and local learning, or as a flow of *data transformation tasks*, such as map, filter, and reduce that can be performed on the data.

A. ML applications as a flow of steps

The deployment process of a machine learning application begins with data selection, where relevant datasets are chosen from various sources based on the goals of the analysis. Subsequently, data preprocessing is conducted to ensure data quality, integration, transformation, and reduction. After data preprocessing, the transformed data is subjected to ML techniques, applying algorithms such as clustering, classification, or association rule mining to extract meaningful patterns. Learning can occur in a variety of ways, such as local by enabling partial learning on device data, aggregated by merging and analyzing data from multiple sources, or global by leveraging large-scale datasets for comprehensive analysis and modeling.

Table I describes which layers of an edge-cloud architecture are most appropriate for executing the steps of a machine learning application:

- At the *device layer*, data can be collected, filtered on the basis of application requirements, and compressed in order to improve storage efficiency, conserve bandwidth, and enhance data transfer speed. Pre-processed data can be used to train local learning models that can later be employed in federated learning tasks. This layer is critical for preserving data privacy, reducing latency, and conserving network bandwidth.
- At the *edge layer*, local models from IoT devices can be aggregated to facilitate collaborative learning and enhance predictive capabilities. By aggregating the local models, valuable insights and knowledge can be derived from larger and more diversified datasets, leading to more accurate and robust predictions. Additionally, at the edge layer, data can potentially be cached, allowing for quicker access and reducing the need for frequent data transfers to the cloud or fog servers.
- At the *fog layer*, data and models from multiple edge devices can be aggregated, consolidating information to gain collective intelligence. Aggregation allows for collaborative learning and the extraction of comprehensive insights. Moreover, meta-learning can be leveraged to improve the learning capabilities of the architecture.
- At the *cloud layer*, thanks to the capabilities that are not available in other layers, large-scale training, advanced analytics, and data storage can be performed. It can be leveraged for global learning by aggregating data and models from multiple sources, allowing for a broader understanding of global trends and patterns.

As a specific example of a machine learning application that can run on the edge-cloud continuum, let's consider an *object detection and tracking* application. *Edge devices* equipped with cameras capture real-time video or image data from the environment. These devices perform initial processing tasks locally, such as object detection algorithms, to identify and track objects of interest within their immediate vicinity. The *edge servers* can train and update local models using the data acquired, properly compressed, from the devices they are connected to. These models typically include machine learning algorithms designed for object detection and tracking. Training can be performed using locally collected data, enabling real-time responsiveness and reducing the need for frequent communication with the cloud. Once trained, the local models on edge devices can perform real-time infer-

Layer	*Steps*	*Example (Object detection and tracking)*
Device layer	Data collection, filtering and compressing, local learning	Devices capture camera data, which can be stored, compressed and processed
Edge layer	Data aggregation, caching and model aggregation	Train and update local models, enabling real-time inference, detecting and tracking objects
Fog layer	Data aggregation, collective learning, meta learning	Hierarchically aggregation of local models
Cloud layer	Large-scale training, advanced analytics, persistent storage, global learning	Global models, long-term analysis, archiving of historical data and coordination of activities

TABLE I: ML applications as a flow of steps distributed on the layers of an edge-cloud architecture.

ence, detecting and tracking objects in the captured video or images, enabling fast and localized decision-making based on the detected objects. The edge devices can periodically transmit relevant information, such as object detections, to *cloud (or fog) servers*. This information can be aggregated with data from other edge servers to gain a comprehensive understanding of the overall environment and improve the accuracy of the whole process. The cloud layer can provide additional computational resources and advanced algorithms for further processing, handling resource-intensive tasks, such as complex object recognition, advanced tracking algorithms, or long-term analysis of object behavior. The cloud layer can also store historical data for retraining models, as well as provide centralized coordination and management of the entire system.

B. ML applications as a flow of data transformation tasks

Machine learning applications can also be defined as a flow of data transformation tasks. For example, following a functional programming approach, an application could be defined as a stream of mapping, filtering, and reducing functions. As an example, a map function can be applied to a set of data to transform each element of that set. After the map function, a filter function can be used to selectively keep only certain elements from the transformed set. Finally, a reduce function can be exploited to aggregate the filtered elements in order to generate the final result.

Before describing which layer of an edge-cloud architecture is most appropriate for performing certain types of tasks, we describe some factors that we believe should be taken into account. First of all, *heterogeneous technologies, hardware, and software stacks* are important factors to consider to try to take full advantage of the real-time processing capabilities of edge devices with the advanced computing power and centralized resources of the cloud. *Data location* is another key factor, aiming to minimize data movement by performing computation as close to the data as possible. If a given node lacks the required computing capabilities, support can be requested from higher-layer nodes. The *geographical distribution* of hardware components needs also to be taken into account, as the physical location of devices can impact communication overhead and algorithm performance. For these reasons, it is crucial to minimize *data movement* whenever possible and, in cases where data transfer is necessary, prioritize the transfer of compact and summarized information. Moving large volumes of data across the network can introduce latency, consume bandwidth, and increase the overall overhead of the system. By minimizing data movement, the system can operate more efficiently and reduce the risk of performance bottlenecks. Additionally, when data transfer is unavoidable, transmitting compact and summary information rather than the complete dataset can significantly reduce communication overhead. Finally, *task scheduling* and *data persistence* are needed to execute tasks in distributed learning algorithms, ensuring load balancing, prioritizing critical tasks, and potentially replicating tasks for improved execution time and fault tolerance. The storage of temporary data on non-persistent components should be considered as it may lead to data loss.

Figure 1 shows the main data transformation tasks of a distributed algorithm and the corresponding layer of an edge-cloud architecture is the most appropriate for performing them. According to the Spark distributed computing framework, tasks that perform transformation operations on data can be divided into two types: narrow transformation and wide transformation. The main difference between these two types lies in the data shuffling behavior and the level of parallelism during execution. Specifically, *narrow transformation* tasks do not involve data shuffling or movement across partitions, as they operate on a single partition at a time, performing operations such as filtering, mapping, and aggregating within the partition. These tasks can be executed in parallel across partitions without requiring communication or coordination between them. On the other hand, *wide transformation* tasks require data shuffling or movement across partitions. They depend on data from multiple partitions and often involve operations like joining, grouping, and sorting, thus requiring coordination and communication between partitions to exchange and reorganize the data, which can result in reduced parallelism and increased overhead.

Considering an edge-cloud architecture, narrow transformation tasks are better suited to be performed on the device or edge layers. Narrow tasks operate on a single partition at a time and focus on local data processing within a specific partition. They involve operations like filtering, mapping, and aggregating, which can be executed in parallel across partitions without the need for extensive communication or

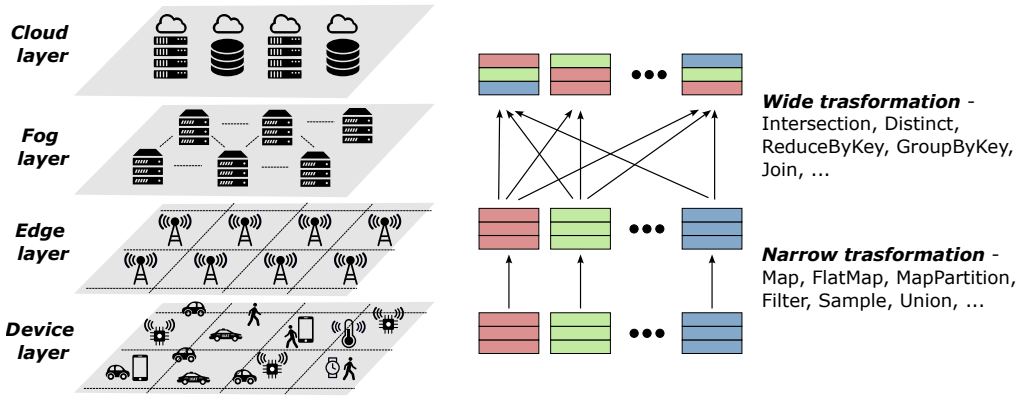

Fig. 1: Tasks of a machine learning algorithm distributed on the layers of an edge-cloud architecture.

coordination between them. Since the data is mainly generated by the devices, it is already naturally partitioned and can be analyzed where it was generated (or combined at the edge level), reducing latency and network congestion and enabling faster response times and enhanced privacy. On the other hand, wide transformation tasks, which involve data shuffling and movement across partitions, are more suitable to be performed at the fog or cloud layers. Wide tasks often include operations like joining, grouping, or sorting, which require coordination and communication between partitions to exchange and reorganize data. In this way, by exploiting the increased computational/storage capabilities and higher scalability, it is possible to handle complex data-intensive operations, facilitate inter-partition communication and enable efficient data exchange and coordination required for this kind of task.

IV. Examples of machine learning algorithms distributed across the edge-cloud continuum

This section describes how two machine learning algorithms, namely k-means and Random Forest, can be suitably adapted to run on the edge-cloud continuum. The task flow of the algorithms is described and, for each task, the most suitable layers of an edge-cloud architecture are identified in which to execute it.

A. K-means

The k-means algorithm [29] is widely recognized as one of the most popular and effective clustering algorithms. It is an unsupervised learning technique known for its simplicity of implementation and usage. The algorithm operates by defining centroids, which are points in the feature space representing the average distance of data points within a cluster. Initially, the algorithm randomly selects centroids and then iteratively refines their positions to optimize the clustering. The process continues until the centroids stabilize, indicating successful clustering, or the specified number of iterations is reached.

The implementation discussed here is Parallel k-means (PKMeans) [13], a parallel version based on the MapReduce paradigm. This version can be implemented as a flow of three main data transformation tasks.

1) The first task of the algorithm is the *map*. Each worker node receives a partition of input data and the initial K centroids. Each node locally calculates the distance between the data point and all centroids, assigning the point to the closest centroid.
2) After the map task, the algorithm proceeds with the *local combine*. In this task, for each centroid, each node calculates the partial sum of all the data points assigned to that centroid. The local combine stage helps reduce the amount of data shuffled between worker nodes, as each node can independently compute the partial sums.
3) Next, the *reduce* task combines the partial sums calculated by different nodes. For each centroid, the algorithm computes the global sum of all the partial sums received from different nodes. Using the global sum and the total count of data points assigned to that centroid, the algorithm calculates the new centroid.
4) The algorithm then enters a repetition phase, iterating the map, local combine, and reduce operations until convergence is achieved.

For efficient execution of this algorithm in edge-cloud continuum architecture, we can follow the approach shown in Figure 2. The map and combine tasks can be executed between the device and edge layers. If the computation of these tasks is done only at the device layer (if the computational and storage power allows it) we can perform direct analysis on the portions of data generated by the devices. If, on the other hand, we move to the edge layer, we are able to carry out analyses on sets of devices that belong to the same edge. The results from these devices/edges can be then combined at fog/cloud layers to calculate the new cluster centroids. This approach can minimize communication between the edge and the cloud and removes the need for direct communication between the devices themselves. The communication flow in this distributed setting involves the transmission of cluster sum vectors from each device/edge to the fog/cloud, and then the transmission of the new centroids back to each device/edge.

While the discussed approach assumes a dataset distributed across multiple devices, it can also be adapted for scenarios where data continuously flows from the devices. In such cases,

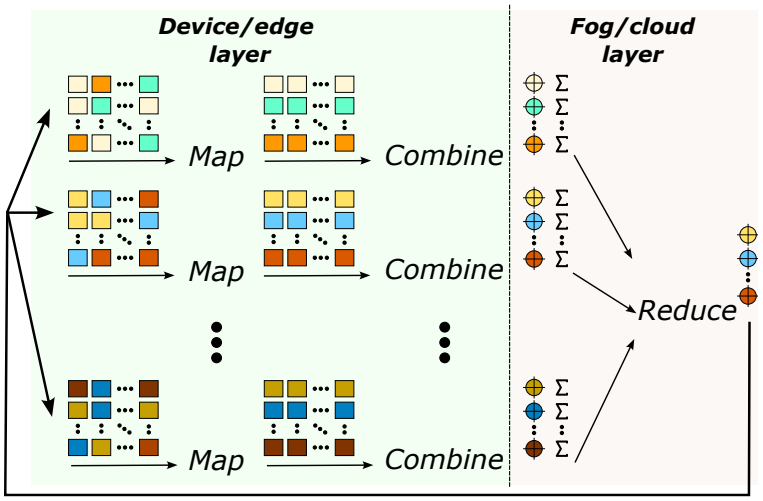

Fig. 2: K-means tasks distributed on the layers of an edge-cloud architecture.

each device can store the data until a sufficient amount is accumulated for training the algorithm. Once the training phase is completed, the returned centroids can be exploited for clustering the newly produced data. By adopting this cloud-edge continuum architecture and leveraging parallel processing, the communication overhead can be minimized, enabling efficient and scalable k-means clustering in distributed environments.

B. Random Forest

Random Forest [30] is a popular ensemble learning algorithm that combines multiple decision trees to make predictions. Each decision tree is built on a random subset of the training data and random subsets of features. The final prediction is made by aggregating the predictions of all the individual trees. To handle large-scale datasets, Random Forest can be distributed across multiple computing nodes in an HPC system. Data is partitioned across the nodes, where a subset of decision trees is built using local data. The predictions from the individual trees are then combined to produce the final prediction. This approach allows for scalable processing and efficient training on large datasets and helps to overcome the memory and processing limitations of worker nodes.

In an edge-cloud continuum architecture, the distributed version of Random Forest can take full advantage of edge-cloud architectures. In fact, by distributing the workload between edge devices and the cloud, it allows to take full advantage of local storage and edge processing capabilities along with the compute power and persistent storage capacity of the cloud. Each edge device can independently train a subset of decision trees using its local data, taking advantage of parallel model training. In this case, if data is not generated uniformly across devices (some devices generate fewer data than others), it could lead to unbalanced training sets and hence decision trees of the forest that are not well trained. For this reason, it is better to combine the trained decision trees from different devices in order to enhance the diversity and robustness of the Random Forest model. This combination can be performed at the edge layer, using data from different devices and exploiting horizontal and vertical randomization techniques to enhance performance and robustness. The dynamic load balancing capability of the edge-cloud continuum architecture can also ensure efficient utilization of resources by offloading computation from overloaded edge devices to the cloud. Additionally, the cloud resources can be exploited for model updates, maintenance, and management tasks, including retraining the model with new data and deploying updated models to edge devices.

This solution can also be adapted for real scenarios where data continuously flows from the devices. In such cases, the model can be updated incrementally as new data arrive, without requiring the entire dataset to be processed again. Instead of training decision trees from scratch, the algorithm can update the existing ensemble by adding new decision trees to the ensemble. This approach allows the Random Forest to adapt to changes in the data distribution over time and incorporate the knowledge gained from new observations.

V. Conclusion

In this paper, we addressed the challenge of efficiently executing distributed machine learning applications and algorithms on an edge-cloud continuum architecture. The main problem lies in the resource constraints of edge devices, the need for real-time analytics, and the requirement for access to large datasets and computational resources for accurate model training. Traditional approaches that rely solely on edge or cloud computing have proven limited in effectively addressing these requirements. To overcome these challenges, we discussed an approach that leverages the strengths of both edge and cloud layers.

Our approach involves breaking down machine learning applications into different levels of granularity. We demonstrated how certain tasks requiring low response times and real-time analytics can be performed at the edge, while resource-intensive tasks, data aggregation, and access to large

datasets are better suited for the cloud. Through the task flow of two machine learning algorithms, namely k-means and Random Forest, we identified which levels of the edge-cloud architecture are best suited for executing each task. In particular, narrow transformation tasks, focused on local data processing, are more suitable for the device or edge layers. In contrast, wide transformation tasks involving data shuffling and coordination across partitions are better suited for the fog or cloud layers.

Future research in this area will further refine and expand upon the capabilities of the edge-cloud continuum architecture, fully unlocking its potential for distributed machine learning. New libraries specifically tailored for machine learning algorithms, by incorporating optimized algorithms and protocols, could further improve performance, scalability, and resource utilization of machine learning applications across the edge-cloud continuum.

ACKNOWLEDGEMENTS

We acknowledge financial support from "National Centre for HPC, Big Data and Quantum Computing", CN00000013 - CUP H23C22000360005, and from "PNRR MUR project PE0000013-FAIR" - CUP H23C22000860006. This work was also supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART") - CUP C37G22000480001.

REFERENCES

[1] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN computer science*, vol. 2, no. 3, p. 160, 2021.

[2] L. Belcastro, R. Cantini, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Programming big data analysis: Principles and solutions," *Journal of Big Data*, vol. 9, no. 4, 2022.

[3] D. Talia, P. Trunfio, and F. Marozzo, *Data Analysis in the Cloud: Models, Techniques and Applications*. Elsevier, October 2015, iSBN 978-0-12-802881-0.

[4] F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Edge computing solutions for distributed machine learning," in *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2022, pp. 1–8.

[5] L. Belcastro, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Edge-cloud continuum solutions for urban mobility prediction and planning," *IEEE Access*, vol. 11, pp. 38 864–38 874, 2023.

[6] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[7] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[8] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A survey on federated learning for resource-constrained iot devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, 2021.

[9] D. Rosendo, A. Costan, P. Valduriez, and G. Antoniu, "Distributed intelligence on the edge-to-cloud continuum: A systematic literature review," *Journal of Parallel and Distributed Computing*, 2022.

[10] C. Savaglio, P. Gerace, G. Di Fatta, and G. Fortino, "Data mining at the iot edge," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–6.

[11] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, pp. 1–16, 2016.

[12] N. K. Alham, M. Li, Y. Liu, and M. Qi, "A mapreduce-based distributed svm ensemble for scalable image classification and annotation," *Computers & Math with Applications*, vol. 66, no. 10, pp. 1920–1934, 2013.

[13] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *IEEE international conference on cloud computing*. Springer, 2009, pp. 674–679.

[14] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 919–933, 2016.

[15] G. Luo, X. Luo, T. F. Gooch, L. Tian, and K. Qin, "A parallel dbscan algorithm based on spark," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud)*. IEEE, 2016, pp. 548–553.

[16] "Apache Mahout," https://mahout.apache.org//, accessed May 2023.

[17] "Apache Spark's MLlib," https://spark.apache.org/mllib/, accessed May 2023.

[18] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 63–71.

[19] Dennis, Don Kurian and Gaurkar, Yash and Gopinath, Sridhar and Goyal, Sachin and Gupta, Chirag and Jain, Moksh and Jaiswal, Shikhar and Kumar, Ashish and Kusupati, Aditya and Lovett, Chris and Patil, Shishir G and Saha, Oindrila and Simhadri, Harsha Vardhan, "EdgeML: Machine Learning for resource-constrained edge devices." [Online]. Available: https://github.com/Microsoft/EdgeML

[20] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2 kb ram for the internet of things," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1935–1944.

[21] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain, "Protonn: Compressed and accurate knn for resource-scarce devices," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1331–1340.

[22] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[23] "MIT Tiny ML project," https://hanlab.mit.edu/, accessed May 2023.

[24] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[25] H. H. Kumar, V. Karthik, and M. K. Nair, "Federated k-means clustering: A novel edge ai based approach for privacy preservation," in *2020 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE, 2020, pp. 52–56.

[26] H. Yao, J. Wang, P. Dai, L. Bo, and Y. Chen, "An efficient and robust system for vertically federated random forest," *arXiv preprint arXiv:2201.10761*, 2022.

[27] Y. Liu, Y. Liu, Z. Liu, Y. Liang, C. Meng, J. Zhang, and Y. Zheng, "Federated forest," *IEEE Transactions on Big Data*, 2020.

[28] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu *et al.*, "Fedml: A research library and benchmark for federated machine learning," *arXiv preprint arXiv:2007.13518*, 2020.

[29] J. A. Hartigan, M. A. Wong *et al.*, "A k-means clustering algorithm," *Applied statistics*, vol. 28, no. 1, pp. 100–108, 1979.

[30] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.