Journal of Big Data

CASE STUDY

Open Access

# Efficiently approaching vertical federated learning by combining data reduction and conditional computation techniques

Francesco Folino[1†], Gianluigi Folino[1†], Francesco Sergio Pisani[1†], Luigi Pontieri[1†] and Pietro Sabatino[1*†]

†Francesco Folino, Gianluigi Folino, Francesco Sergio Pisani, Luigi Pontieri and Pietro Sabatino have contributed equally to this work.

*Correspondence: pietro.sabatino@icar.cnr.it

[1] ICAR-CNR, Via P. Bucci, 8/9c, Rende 87036, Italy

## Abstract

In this paper, a framework based on a sparse Mixture of Experts (MoE) architecture is proposed for the federated learning and application of a distributed classification model in domains (like cybersecurity and healthcare) where different parties of the federation store different subsets of features for a number of data instances. The framework is designed to limit the risk of information leakage and computation/communication costs in both model training (through data sampling) and application (leveraging the conditional-computation abilities of sparse MoEs). Experiments on real data have shown the proposed approach to ensure a better balance between efficiency and model accuracy, compared to other VFL-based solutions. Notably, in a real-life cybersecurity case study focused on malware classification (the KronoDroid dataset), the proposed method surpasses competitors even though it utilizes only 50% and 75% of the training set, which is fully utilized by the other approaches in the competition. This method achieves reductions in the rate of false positives by 16.9% and 18.2%, respectively, and also delivers satisfactory results on the other evaluation metrics. These results showcase our framework's potential to significantly enhance cybersecurity threat detection and prevention in a collaborative yet secure manner.

**Keywords:** Vertical federated learning, Mixture of experts, Deep learning, Green AI, Cybersecurity

## Introduction

Traditional centralized machine learning (ML) models and methods encounter data privacy and security challenges when applied to sensitive/private data, which abound in many real-life application domains like cybersecurity and healthcare. Legislation like the European Union's *General Data Protection Regulation* (GDPR), introduced in 2016, poses strong limits to collecting and publishing citizens' data. Moreover, several recent episodes of data leakage over the Internet have made people and organizations increasing concerned about the privacy of their data. On the other hand, there's a growing need for sophisticated data analysis services driven by AI models, which require large amounts of data and expensive computations. Data has now emerged as a precious resource that citizens/organizations are reluctant to share, while the computational

resources required by *Deep Learning* (DL) systems result in both increased operational costs and a significant energy consumption and pollution footprints. *Federated Learning* (*FL*) has emerged as a promising solution paradigm for addressing both these contrasting needs by allowing multiple parties to train a global model collaboratively without requiring them to share their raw data. In principle, this paradigm has the potential of enabling the discovery of accurate models from distributed data while avoiding the disclosure of sensitive/private information.

However, FL solutions can be particularly demanding in terms of network bandwidth, memory, computation power and energy, especially when dealing with large amounts of data or large-scale computer networks, due to the need to perform many data processing and communication operations (involving raw data, intermediate results and gradients). In fact, the swift advancement of AI is driven by progressively more extensive and computationally intensive machine learning models and datasets. Consequently, the computational resources used in training cutting-edge models are escalating exponentially, doubling every ten months from 2015 to 2022, resulting in a substantial carbon footprint [1]. Thus, if FL (combined with secure aggregation and differential privacy methods [2]) can support complex ML tasks in a collaborative, decentralized and privacy-aware manner, its associated communication and computation overheads may well lead to significantly higher carbon emissions than centralized solutions. Indeed, recent research showed that training a model through FL may result in up to 80 kgs of carbon dioxide equivalent (CO2e), surpassing the emissions from training a larger transformer model in a centralized setting using AI accelerators [3]. This inefficiency stems from factors such as training overhead across diverse client hardware, added communication costs, and slower convergence. The global carbon footprint of FL is expected to rise with increased industry adoption and the shift of ML tasks from centralized settings. This concern is amplified by the limited availability of renewable electricity in various locations, posing a challenge to achieving environmentally friendly FL [4]. Seizing opportunities for efficiency optimization in FL is crucial to promoting greener ML applications.

Considering the widely reckoned urgency of curbing the energy impact and carbon footprint of ML applications [5], it comes with no surprise the recent proposal of green approaches to FL [1, 6, 7] aimed at suitably trading off energy efficiency and performance. Reducing AI's environmental impact, emphasizing energy-efficient algorithms, hardware development, and data center carbon footprint reduction are some of the main goals of Green AI [5]. While renewable energy can power centralized AI, providing FL with renewable energy is challenging due to end-user devices tied to local energy mixes. Unfortunately, most of the approaches proposed to far, at the border between Green AI and FL, mainly focus on energy-aware node selection and job assignment strategies, and they do not fit the challenging emerging setting of *Vertical Federated Learning* (VFL), in which multiple parties store data concerning different feature spaces but overlapping real-world entities, while possibly having the true labels reside on a third-party server .[1]

[1] Consider, as an instance, a collaboration scenario involving a credit bureau, an e-commerce entity, and a bank aiming to create a model for predicting user credit scores. The credit bureau possesses exclusive access to users' credit data and requires rigorous privacy safeguards for data-owner client nodes. These nodes are responsible for managing and processing their private data locally and must interact with the coordinator node at each optimization step during training, deviating from conventional Horizontal Federated Learning (HFL) methods. In such cases, Vertical Federated Learning (VFL) emerges as a more appropriate option. However, it is worth noting that the potential scope of VFL frameworks goes beyond such cross-silo application scenarios, and can embrace cross-device FL applications in the fields of IoT and Edge computing.

Recently, *Mixture of Experts* (MoE) architectures gained considerable attention as an effective means for balancing model capacity, computational cost and energy efficiency. In general, classical MoE models [8] look like an ensemble of homogeneous prediction sub-nets, named *experts*, where a final prediction is obtained for any data instance by linearly combining those returned by the experts with weights that are dynamically computed by a further sub-net, named *gate*, based on the data instance. In order to enable efficient conditional computations, in *Sparsely-Gated Mixture-of-Expert* (SMoE) [9–12], the gate sub-net is made to only select a fixed small number $k$ of experts (the ones that look the $k$ most competent ones for the current input data instance). In principle, sparse MoE models enjoy two nice properties that make them a promising solution for green FL applications: (i) by physically deploying and training all the expert models in the nodes where the privacy-sensitive data are kept, one can reduce the risk of exposing these local data and models to information leakage during the distributed training procedure (which only requires each node to share information on the predictions that its expert is making on its local data rather than the model's parameters, the raw data or some embedded representation of the latter); (ii) the conditional computation paradigm supported by a sparse MoE architecture allows for reducing the cost of data transfers when a batch of data instances is to be classified, since this service can be provided efficiently by exploiting the gate sub-model to select a very small number (e.g., just one or two, as explored in the experimental study of Sect. 5) of expert sub-models for each of these instances, group the instances based on the the experts that have been chosen for them, route these data groups to the nodes hosting the chosen experts, and eventually gather the resulting expert predictions for each group.

*Research gap* Recent research has addressed the problem of analyzing [3, 4, 13] and/or reducing [14–24] the energy demand and carbon footprint of FL applications [1]. However, most of this research has focused on HFL settings, paying no attention to the challenging case of VFL applications, where the parties are obliged to cooperate tightly across all training iterations by exchanging derived information concerning the parameters/gradients of their local models. To prevent information leakage, secure communication protocols have been proposed that involve many peer-to-peer communications and encryption-based data transformation [25].

Despite these above-mentioned potentialities of sparse MoEs for FL applications, only a few MoE-based FL methods have been proposed so far, which mostly address the sole HFL scenario [26–29], typically with the aim of dealing with heterogeneous data distributions and/or model personalization. In fact, to the best of our knowledge, the idea of exploiting a MoE-based model in a VFL setting has only been proposed in [30]. However, the technical solution defined in [30] suffers from a number of drawbacks in terms of information leakage risks and communication/computation demand since it relies on using embedded versions of the raw local data of the participating nodes obtained with the help of an auto-encoder model, to train the gate sub-model in the coordinator node that is driving the learning process. Indeed, transmitting these (potentially) large data embeddings may lead to significant communication overheads, while the embedded local datasets themselves are exposed to "inversion attacks", compromising data privacy. To mitigate this risk, each local node could implement more sophisticated encryption methods based, e.g., on differential privacy [31] or homomorphic encryption [32].

However, while enhancing privacy, these methods also increase computational and communication costs, highlighting the critical trade-off between privacy and efficiency in FL systems.

*Contribution* To address the above-described gap of research in the search for a VFL method able to find a satisfactory balance between the needs of training an accurate prediction model and of curbing the computation/energy costs and related carbon footprint while preserving data privacy, this paper introduces a novel and efficient MoE-based approach to VFL (named `VFL_MoE`), which supports scalable and efficient computations without compromising data confidentiality.

Differently from [30], in the proposed approach, the gate is trained by leveraging only a smaller fraction of (possibly sanitized) data features that are ensured not to be privacy-sensitive (e.g., raw features encoding generic patients' information that does not disclose their medical conditions or diseases, or sanitized features) and safely shared across the nodes. Avoiding the transmission of embedded data allows for curbing communication overheads, in addition to reducing privacy pitfalls.

To further lower the computational burden, the proposed approach also allows for controlling the fraction of data employed per training epoch through a data-reduction factor $r$, according to the *Repeated Random Sampling* (RRS) strategy proposed in [33]. More specifically, the approach employs a subset of the data batches in each training epoch as an efficient alternative to conventional data pruning/distillation methods that would entail additional computational efforts in a pre-processing stage.

In summary, the main contributions of this proposal are the following:

- A MoE-like model architecture for Vertical Federated Learning, which minimize the exposure to leakage risks of private local data in the client nodes by requiring the latter to only provide the coordinator with the (scalar) per-instance output returned by their respective expert models, without the need of sharing their raw data instance or learnt/embedded representations of them.
- A distributed algorithm for training a model following this architecture in accordance with privacy-preservation and cost-reduction requirements that arise in typical VFL scenarios. The algorithm allows for keeping under control both communication and computation costs through a data-reduction hyper-parameter $r$, according to an RRS-based strategy.
- A theoretical analysis of the computation and communication costs of the approach, and an experimental study of how the accuracy of the model discovered depends on both factor $r$ and the number $k$ of experts selected by the gate. This analysis, offering in-depth insights into the model's performance under different operational settings, showcases the effectiveness of the proposed framework in terms of model accuracy when using small values of $r$ and $k$, so achieving a satisfactory trade-off between model performance and energy saving.

The novelty of this research work is discussed in more detail in Sect. 6, which compares the proposed framework to previous research work in the field.

*Organization* The paper's structure is organized as follows: Sect. 1 initiates the discussion by addressing data privacy challenges in the machine learning domain, focusing on

Federated Learning (FL) and its inherent limitations. This section also delves into the concept of Green AI, highlighting its growing significance in the contemporary machine learning field, especially concerning energy efficiency and environmental implications. Section 2 explores Vertical Federated Learning (VFL), covering critical aspects of data sovereignty, privacy concerns, and the nuances of the Mixture of Experts (MoE) model. The paper then progresses to Sect. 3, where a formal framework for the VFL challenge is articulated. In Sect. 4, a novel methodological approach is unveiled, detailing the unique model architecture and the training algorithm. Section 5 is dedicated to empirically validating the proposed approach through comprehensive experiments, benchmarking it against existing methods. A thorough review of the current state of pertinent literature is provided in Sect. 6. Finally, Sect. 7 concludes the paper with a summary of the proposed approach and insights into potential avenues for future research.

## Background

### Vertical federated learning and privacy

In recent years, the advent of data-driven technologies has prompted a paradigm shift in how data is processed, analyzed, and utilized across various domains. However, this progress has been accompanied by growing concerns over data privacy, security, and the sovereignty of personal information [34].

Indeed, data sovereignty [35], in the context of digital data, focuses on the owner's authority over these data. It goes to the extent of specifying which aspects of the data the owners are willing to share and with whom. At the European level, particularly in the digital realm, Data Sovereignty encompasses two primary domains: Cloud Sovereignty, involving the adoption of federated cloud services and infrastructures that comply with existing regulations, and the secure online exchange of data among multiple participants in a consortium or group of companies [36].

In addition, formal contracts that govern the usage and access to data, outlining how data can be shared with other entities or organizations, must be introduced and handled [37], also including business, legal, and cloud-based regulations. To illustrate, consider a scenario where a health operator, acting as both a producer and consumer of data, provides certain data and utilizes the outcomes of analytics/machine learning operations. In this complex scenario, various operations need to be carried out, and depending on the operator's role (patient, paramedical staff, or medical doctor), not all data features may be accessible. Furthermore, certain data elements may need anonymization, and restrictions may apply to transferring data outside the country of origin or the European Community. To handle such a scenario adequately, the federated learning paradigm [38] has emerged as a compelling solution to reconcile the benefits of data-driven insights with the imperative to safeguard data sovereignty.

Federated Learning (FL) is a decentralized ML approach that allows models to be trained across multiple decentralized devices or servers holding local data without exchanging the raw data itself. By allowing model training to occur locally on devices or servers that house the data, federated learning minimizes the need for raw data transfer. In addition, FL supplies decentralized training, enabling model training to occur locally on distributed devices, preserving data privacy while collectively improving the model's performance and guaranteeing secure aggregation by updating the models from multiple

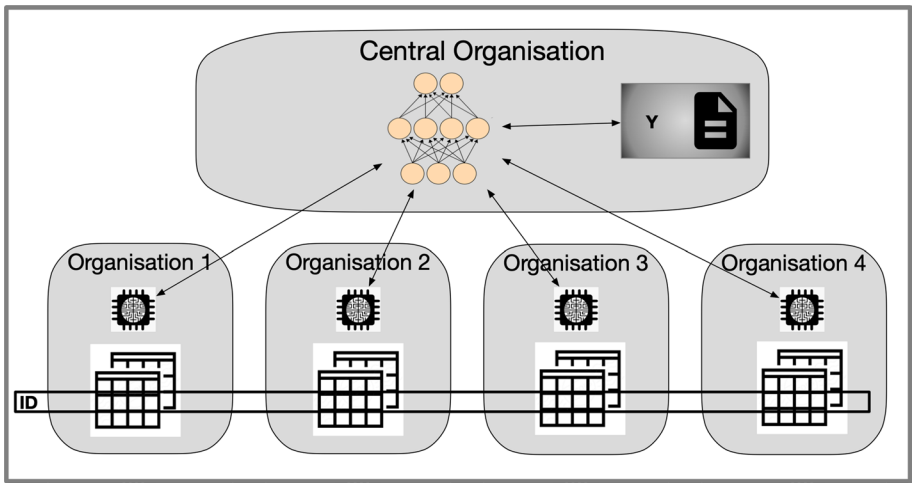


**Fig. 1** A vertical federated learning scenario where both the model to be trained and the training data are split among different organisations

sources without exposing individual contributions, ensuring confidentiality and privacy. Another mechanism that can be included is differential privacy, which adds noise to individual updates to protect against identifying specific data points, contributing to a robust privacy framework.

While FL holds great promise, challenges like communication overhead, non-IID (non-identically distributed) data, and security concerns persist. Ongoing research focuses on addressing these challenges to enhance further the applicability and robustness of federated learning in diverse environments, including the recent adoption of vertical federated learning frameworks [39].

In the context of Vertical Federated Learning, data undergo partitioning based on features across distinct parties. The primary objective is to enable these parties to construct a prediction model collaboratively while protecting sensitive data from being divulged to other participants. In contrast to Horizontal Federated Learning, the VFL setting necessitates a more intricate mechanism for decomposing the loss function at each party.

Two different strategies [40] are usually followed: a) the model is owned by each party participating in the training; b) the model is split among the different parties. Typically, in the latter case, as illustrated in Fig. 1, each node transforms its input data into an intermediate data representation (as the output of a hidden layer of a classic neural network). This intermediate data is transmitted to the next segment until the training or the inference process is completed. During the backpropagation procedure, the gradient is also propagated across the different nodes. That can also be useful to reduce the computational burden of each node, which in many real-world scenarios may have limited computational resources.

#### Mixture of experts (MoE) classifiers

Recent advancements in Machine Learning have underscored the significance of model scaling to enhance and deploy real-world machine learning applications [41]. The substantial success of deep learning across various fields, including natural

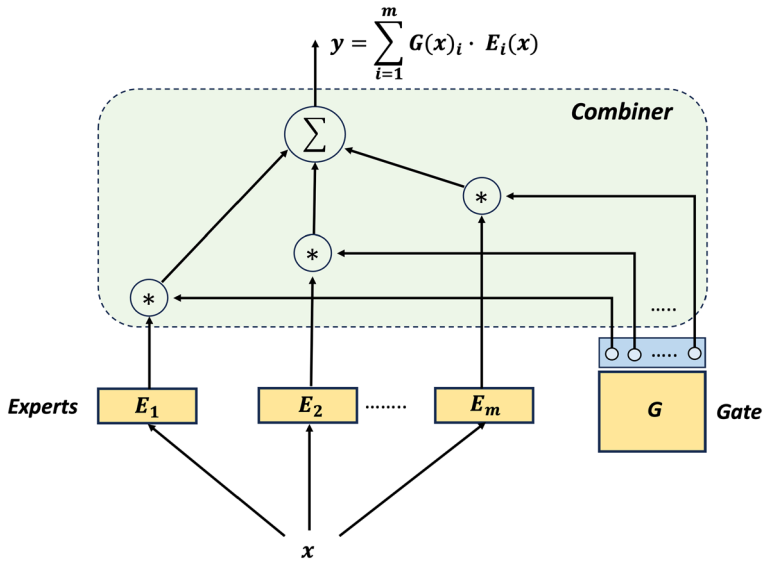


**Fig. 2** Illustration of the standard Mixture of Experts (MoE) architecture comprising *n* experts. Each subnet $E_i$, denoting an expert, computes a unique classification function based on the input *x*. The gating function *G* assigns per-expert competency scores denoted by $G(x)_1, \ldots, G(x)_m$, which sum up to 1. The final output *y* is a weighted sum of the experts' outputs, weighted according to the competency scores determined by *G*

language processing, computer vision, and audio analysis, can be attributed mainly to scaling both the model size and the training data [42–44]. However, the associated computational cost grows quadratically with model size, outstripping the pace of hardware advancements and posing sustainability challenges.

Therefore, as machine learning models become progressively larger to capture the complexity of real-world data, the quest for computational efficiency has led to innovative architectures that balance model capacity and computational cost. A promising solution to this issue is the *Mixture of Experts* (MoE) framework, which enables model scaling without entailing increases in computation [8]. MoE achieves this through a modular neural network architecture where subsets of the network, known as experts, are activated conditionally depending on the input [45].

The MoE framework (c.f. Fig. 2) and its sparser variant, the *Sparsely-Gated Mixture-of-Expert* (SMoE), embody this balance through *conditional computation*, an approach that dynamically adapts the active model components to the input [9].

Conditional computation provided by MoE models facilitates a dynamic and input-dependent model sparsity, which contrasts starkly with the static sparsity patterns induced by traditional weight pruning techniques [46]. Where static approaches permanently remove weights to reduce parameters and computation, MoE models retain all parameters while selectively activating subsets of them, leading to a versatile model that adapts to varying data regimes [10–12].

The gating mechanism in a MoE model is the critical component that embodies this principle of conditional computation. It directs the flow of inputs to the appropriate experts, separate neural network modules specialized in different regions of the input space, which can be mathematically expressed as:

$$G(x) = Softmax\left(Z(x)\right) \tag{1}$$

where $x$ is the input representation and $Z$ is an MLP network returning the logits of the experts' competency weights in the form of a vector of non-normalized real-valued competency scores. The Softmax function ensures a probabilistic interpretation of these gating scores, trying to promote a sparse activation pattern at the same time.

In the specific case of SMoE models, the sparse gate network $G$ processes input tokens $x$ and computes a distribution over the expert networks, formulated as:

$$G(x) = Softmax(Top\left(Z(x), k\right) \cdot Z(x)) \tag{2}$$

where function $Top(v, k)$ returns a vector of the same size of vector $v$ containing a value of 1 in the positions of it that correspond to the $k$ greatest values in $Z(x)$, and 0 in all the remaining positions. This allows the gate to implement a sparse strategy [10–12] for selecting which experts must classify $x$.

As for the experts, each expert $E_i$, parameterized independently, contributes to the final output based on its gating score $G(x)_i$. The output of the MoE/SMoE is a weighted sum of the experts' outputs, with the weights determined by the gating network $G$:

$$y = G(x)^t[E_1 \dots E_m]^t = \sum_{i=1}^{m} G(x)_i \cdot E_i(x)$$

where $m$ is the total number of experts, and superscript $t$ stands for the matrix/vector transpose operator.

It is worth noting that training a MoE model is a non-trivial endeavour due to the complexities involved in learning the gating function alongside the expert networks. The challenges are several, from the need for efficient back-propagation algorithms that can handle the sparse activations [47] to the potential unbalanced load distribution across experts, which can result in a small subset of experts dominating the learning process [10]. To mitigate these issues, recent approaches have introduced novel training strategies, such as auxiliary load-balancing losses, which encourage a more uniform utilization of experts [10, 11]. Furthermore, the initialization and joint training of gates and experts is crucial to avoid the pitfalls of random initial routing and the long convergence times associated with reinforce-based updates [12, 48]. Other approaches include using gradient approximation methods [49], which can reduce the computation overhead.

SMoE models, in particular, have demonstrated the potential for efficiently scaling model capacity with a constant computational budget by exploiting the sparsity in expert activations. This efficiency is achieved without compromising the representational power of the network, making SMoE models particularly appealing for applications where computational resources are a limiting factor.

In conclusion, MoE models, particularly with sparsity, represent a paradigm shift towards more scalable and computationally efficient neural network architectures. They provide a tractable means of managing model complexity and capacity, paving the way for continued growth in the performance of ML systems without disproportionately increasing the computational overhead.

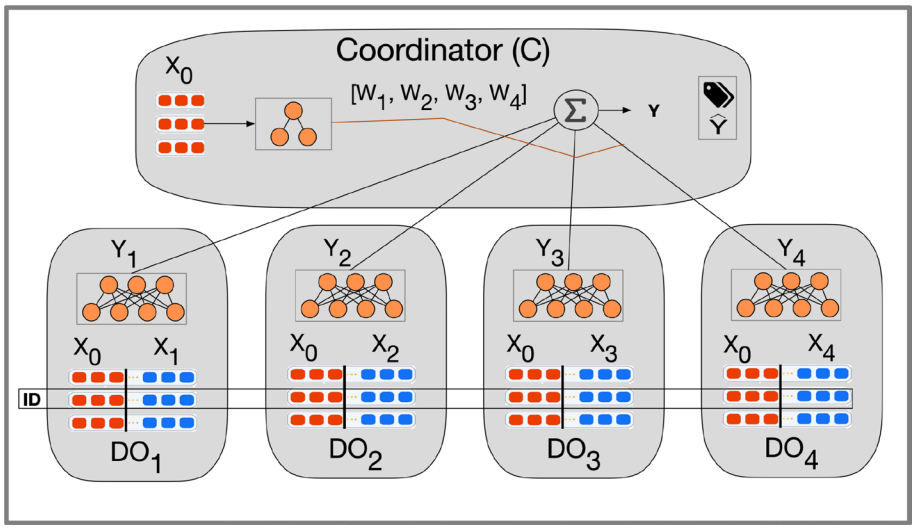


**Fig. 3** Architecture of our MoE-based SVFL framework, in the case where there are $m = 4$ data owners

## Formal framework and problem statement

Let us consider the scenario sketched in Fig. 3, as a variant of typical Vertical Federated Learning (VFL) settings.

In this scenario, an arbitrary number, say $m$, of parties, named *Data Owners* (*DOs* for short), are willing to collaboratively train a classification model under the supervision of a *Coordinator* party ($C$ for short) —in VFL literature, the data owners and coordinator have also been referred to as *passive* parties (or clients) and *active* party (or aggregator), respectively. Assuming that each party in this VFL scenario corresponds to a distinguished node of a computer network, for the sake of generality, let us use the terms node and party as synonyms hereinafter when referring to either a data owner (DO) or the coordinator (C).

In more detail, the data owners, denoted hereinafter as $DO_1, \ldots, DO_m$, store different shards of a dataset $D \subseteq \mathscr{D}$, which keep information about different subsets, say $\mathscr{A}_1, \ldots, \mathscr{A}_m$ of features for the same collection of data instances, where $\mathscr{D}$ denotes the whole data instance space/universe/population $D$ belongs to.

Let us assume that the feature set $\mathscr{A}_s$ of data owner $DO_s$, for $s \in [1..m]$, consists of the union $\mathscr{A}_s = F_0 \cup F_s$ of two feature subsets: a fixed subset $F_0$ of globally shared (public or privacy-safe) data features and a DO-specific subset $F_s$ of local features that convey sensitive information and hence need to be kept private.

Besides storing a copy of the values taken by the shared features $F_0$ on all the data instances, coordinator $C$ also stores the ground-truth class labels of the data instances, regarded as the collection of values that a distinguished target (class) feature $Y$ takes over the data instances.[2]

[2] The above assumption on $F_0$ naturally fits many real-life contexts —for example, both healthcare service providers and insurance companies likely store general information on their customers (concerning, e.g., their gender, age, county/province) that could be safely shared without any risk of disclosing the patient's identity or private information of her. Notably, this assumption does not affect the generality of our problem setting and the applicability of the proposed solution approach: should there be no predefined subset $X_0$ of public features, the parties might well agree on sharing —prior to starting engaging their VFL cooperation— data concerning a (possibly perturbed/embedded or even encrypted) small subset of features that do not pose privacy leakage risks.

For the sake of presentation and of simplicity, let us also assume that all the parties have consistently mapped each of their data instances to the same instance identifier, represented hereinafter as an index (i.e. a natural number) in $[1..N]$, where $N = |D|$ is the number of data instances in the dataset.[3]

Let us regard each data feature $a$ in $\cup_{s\in[0..m]}F_s$ as a function of the form $f : \mathscr{D} \rightarrow \mathbb{R}$ mapping any data instance $d \in \mathscr{D}$ to a real value —for the sake of presentation and w.l.o.g., we are here assuming that all the data instances have been preliminary put into a vectorial form by using any of the many numericalization methods available for training/applying Neural-Network models.

Let us also assume that a specific labelling function $\xi : \mathscr{D} \rightarrow \mathbb{R}^{|\mathscr{C}|}$ exists that returns a one-hot-encoding representation $\xi(d^i)$ of the ground-truth class of any data instance in $\mathscr{D}$, where $\mathscr{C}$ denotes the set of the instance classes.

For any data instance $d^i$ of the given dataset $D$, with $s \in [1..N]$, let $id(d^i) \equiv i$ be the identifier of $d^i$, $y^i = \xi(d^i)$ be the class label of $d^i$ (known only to node $C$), $x^i = [(x_0^i)^t (x_1^i)^t \ldots (x_m^i)^t]^t \in X_0$ be the input-feature vector of $d^i$, where superscript $t$ stands for the matrix transpose operator, while $X_s$ and $x_s^i$ are the feature sub-space induced by feature set $F_s$ and the feature vector representing $d^i$ in sub-space $X_s$ (i.e., $x_s^i = \oplus_{a\in F_s}[a(d^i)]$, with $\oplus$ standing for vector concatenation) for any $s \in [0..m]$ —please remember that a copy of $x_0^i$ is kept by all the VFL parties, including $C$, whereas $(x_s^i)$ is only stored locally to the node of $DO_s$ for all $s \in [1..m]$.

Based on the concepts and notation introduced so far, we are in a position to formally state, in a general form, the collaborative classification task that our research work was aimed at supporting.

**Definition 1** (VFL Problem) Given: a VFL network, keeping information on a dataset $D$ and featuring a coordinator node $C$ and $m$ DO nodes $DO_1, \ldots, DO_m$, such that: (i) each DO node $DO_s$ stores a projection of $D$ over both a proprietary feature space $X_s$ and a shared feature space $X_0$ (corresponding to some feature sets $F_s$ and $F_0$, respectively); (ii) $C$ stores the class labels $y^i$ and the feature vectors $x_0^i$ for all the data instances in $D$ (constituting a partial class-annotated view of $D$ over the shared feature space $X_0$). Find (in a collaborative distributed way): a *Federated Classification Model* (*FCM*) $\mathscr{M} : \mathscr{D} \rightarrow \Delta_m$ (with $\Delta_m$ denoting the probability simplex over $\{1, \ldots, |\mathscr{C}|\}$) that can map any instance of the instance space $\mathscr{D}$ to a discrete probability distribution over the classes. □

In general, an FCM $\mathscr{M}$ could be exploited to support the classification of (novel) data instances in both node $C$ and any data-owner node. However, in a setting where the ground-truth class labels of the training data are regarded as sensitive information of $C$, one can prevent all the other parties from using $\mathscr{M}$, and make the latter available to $C$ only.

As usual in FL contexts, when discovering $\mathscr{M}$, different nodes of the VFL network are expected to store and process information on model parameters in a distributed manner.

[3] In fact, to solve the problem of matching data instances across different data shards, one can resort to a wide range of consolidated solutions developed in the area of Data Integration (as already done in previous VFL applications). Abstracting from this problem allows us to concentrate on the core research goal of devising an effective and efficient methodology for training a classification model in a VFL scenario like the one described above.

However, this distributed computation process must adhere to the following fundamental privacy constraint: the raw data and local model parameters of all the data owners $DO_1 \ldots, DO_m$ cannot be disclosed or exposed to non-negligible risks of information leakage.

As discussed before (see Sects. 1, 2 and 6), previous proposals addressing the VFL problem presented above have mainly focused on security and privacy-preservation issues while paying no or minor attention to computation/communication costs and energy/resource demands. Thus, most of the extant solutions in this field incur high compute burdens, owing to one of the following reasons or to a combination of them: **(1)** In general, training and test procedures in VFL settings need more communication and data processing operations (involving, e.g., the computation and exchange of intermediate results or of gradients) than HFL ones. This tends to make VFL particularly demanding (in terms of network bandwidth, memory, computation power and energy), especially in the presence of large datasets, wide-scale computer networks or resource-constrained nodes. **(2)** The approaches relying on exchanging local raw data and/or model parameters resort to robust information encryption methods to impede leakage of private local information. This leads to neatly increased communication and computation costs compared to a (naive) encryption-free distributed elaboration of the same data. **(3)** The heterogeneity (in terms of distribution, types and quality) of the data involved in the training process may lead to instability and convergence issues, so requiring additional training epochs to meet satisfactory/acceptable accuracy performances.

Similarly to previous VFL approaches, we regard the FCM as a composition of different smaller sub-models, one for each node in the FL network. In particular, each DO node maintains a sub-model focusing on the raw data shard of the DO, whereas a distinguished combiner/aggregator sub-model is responsible for deriving an overall classification output out of the intermediate outputs returned by the other sub-models. For the sake of simplicity, we assume that the combiner sub-model is deployed in the coordinator node C (which owns the class labels of the training data and can hence directly evaluate the model quality along the whole training procedure). Anyway, for the sake of improved security, this VFL architecture can be extended by introducing a separate node playing as a secure, trusted interface between the (active) party that wants an FCM model and the (passive) parties that own different vertically-partitioned shards of the dataset.

Essentially, in our approach, the coordinator node is responsible for leading the training of the FCM, as well as for implementing a classification service based on the FCM .[4]

When processing (the vectorial representation of) any data instance $x^i$ (at either training or test times), in each forward step, the intermediate outputs produced by the local models (residing in the DO nodes) are taken as input by the combiner (residing in the coordinator node) to produce a final class-prediction output for $x^i$. Moreover, in each optimization step of the training procedure, for each parameter $w$ of a local model kept by a data-owner node $DO_s$, the loss gradient of $w$ is propagated from (the combiner

[4] A typical use case of such a model occurs when an application running on the coordinator node (typically on behalf of a suitably authorized client running on another node of the intranet) needs to classify a novel data instance $x'$ that was not already used in the training step.

model residing in) the coordinator node back to the node $DO_s$, to allow the latter to update parameter $w$ optimally.

## The proposed VDL methodology: model architecture and training algorithm

In what follows, a novel methodology is presented for solving the VFL problem introduced in the previous section (see Def. 1). Compared to existing VFL approaches, the methodology aims to better trade-off between the diverging objectives of maximizing model accuracy and minimizing computational costs while ensuring a satisfactory level of privacy for the local raw data of the data owners.

The methodology consists of two major components, namely a MoE-like architecture for the *FCM* classification model to be discovered and a distributed training algorithm, which are presented next in two separate subsections.

### Model architecture

Our approach to training an FCM (i.e. a classification model addressing the reference VFL problem of Def. 1) is based on using a MoE-like model architecture, named hereinafter *VFL_MoE*, along with a distributed training algorithm.

The proposed *VFL_MoE* model (formally specified later on in Def.2) specializes the classical abstract MoE model (sketched in Fig. 2 and described in Sect. 2.2 to our problem setting, suitably addressing the privacy and scalability issues characterizing real-life VFL applications.

Conceptually, the architecture of a *VFL_MoE* is pretty similar to that of a classical neural-net MoE classifier (like that in Fig. 2), in that it is composed of the following functional components (all implemented as different parts of a comprehensive neural-net model): *(i)* Multiple "expert" classifiers $E_1, \ldots, E_m$, each of which can return a (class) prediction for any novel data instance $x$; *(ii)* a *combiner* module consisting of a trainable *gate* sub-net $G$ and a fixed product-sum sub-net implementing a linear convex combination scheme similar to the one sketched in Fig. 2, where the prediction of each expert is weighed through a normalized competency score assigned by $G$ —ensuring that the more competent an expert, the more it leads the overall prediction. In particular, as in sparse MoEs, all the weights returned by the gate are zeroed but the highest $k$ ones so that the product-sum combination reduces to just returning the prediction made for $x$ by the (apparently) best $k$; this is obtained by applying the sparse-gating computation shown in Eq. 2.

However, the combination scheme proposed in this work differs from that of traditional MoEs and sparse MoEs in two key respects:

- For the sake of both privacy and computation efficiency, when classifying any the vectorial representation $x \in X_0 \times X_1 \times \cdots \times X_m$ of any data instance, the gate is made to estimate the competency weight of the experts by only using the partial representation $x_0$ of $x$ related to the common data features shared among all the parties of the VFL network.
- To curb computation and communication costs, in the training process, the gate network is encouraged to be as selective as possible by using an ad hoc training loss function (see Eq. 3, explained later on).

For the preservation of information privacy and computational efficiency, at a physical level, the *VFL_MoE* model is partitioned into several sub-networks allocated to distinct nodes of the VFL network, as illustrated in Fig. 3 and delineated below:

- Every expert is fully allocated to a single data owner (DO) node (at least in the training process), and it is made to focus only on the subset of data features available in that node. This solution allows the expert to be trained on raw data directly –without resorting to any data embedding/encryption mechanism that would increase both the computation costs and the risk of losing relevant information.
- The gate sub-net $G$ (more precisely, the whole combiner sub-model, also including the non-trainable product-sum module) is maintained in the coordinator node $C$, where the ground-truth class labels of the data instances needed in the training process are stored. For (the vectorial representation of) every input data instance $x^i \in \mathscr{D}$, the gate just takes the sub-vector $x_0^i$, representing the projection of $x^i$ onto the shared feature sub-space $X_0$. Since, for any data instance of $D$, a copy of this sub-vector is available in node $C$ (and in all the other nodes of the federation), there is no need to provide the gate (and hence the coordinator party) with information concerning the private data of the DO/client nodes.

In the following, we focus the analysis on a binary classification setting (i.e., a setting where there are only two classes in $\mathscr{C}$ to be discriminated), while pinpointing that extending our research to the general case of multi-class classification is trivial.

Under this working assumption, the functional and physical architecture of the proposed federated classification model can be formally defined as follows:

**Definition 2** [*VFL_MoE*]

Let *Fed* denote the computer network of a VFL federation, consisting of $m+1$ nodes, including a coordinator node $C$ and multiple data-owner (DO) nodes $DO_1, \ldots, DO_m$. Let $\mathscr{D}$ be the instance universe that data available in *Fed* refer to, $X_0, \ldots, X_m$ be the space corresponding to the disjoint sub-sets $F_0, \ldots, F_m$ of data features, such that $F_0$ are the *Fed*-wide shared features and $F_s$ are the private ones of $DO_s$, for $s \in \{1, \ldots, m\}$. Then, for any chosen $k \in \{1, \ldots, m\}$, a $k$-sparse *VFL_MoE* model for *Fed* is a neural net $\mathscr{N} = \langle \mathscr{N}_g, \mathscr{N}_1, \ldots, \mathscr{N}_m \rangle$ assembling two kinds of neural nets:

- *experts* $E_1, \ldots, E_m$, physically-deployed in the data-owner nodes $DO_1, \ldots, DO_m$, respectively, which encode the local classification functions $\sigma(f_1), \ldots, \sigma(f_m)$ such that each $f_s$ is a real-valued function of the form $f_s : X_0 \times X_s \rightarrow \mathbb{R}$ and $\sigma(\cdot)$ is the standard Sigmoid function.[5]
- a *gate* $\mathscr{N}_g$, physically-deployed in the coordinator node $C$, which encodes a routing function $g \in \Delta_m^{X_0}$ (where $\Delta_m$ is the $m$-dimensional probability simplex) mapping the

[5] This allows each expert to map (the local representation of) any data instance in $\mathscr{D}$ to an estimate of the probability that it belongs to the second class.

shared partial representation $x_0$ of any data instance $d \in \mathscr{D}$ to a discrete probability distribution over the (indexes of the) experts.

Model $\mathscr{N}$ as a whole encodes a classification function $f : X \rightarrow [0,1]$ defined as follows: $f(x) \triangleq \tilde{g}(x_0)^t[f_1(x_1), \ldots, f_m(x_m)]^t$, where $X = X_0 \times X_1 \times \ldots \times X_s, x = [x_0, \ldots, x_m]^t \in X$ and $\tilde{g}(x_0) = Softmax\big(Top(g(x_0), k)\big)$ is the top-$k$-adaptation of $g(x_0)$ (see Eq. 2 and related comments for more details), while $x_0, x_1, \ldots, x_m$ are the feature-vector representation of $d$ in the DOs' sub-spaces $X_0, X_1, \ldots, X_m$, respectively. □

In principle, the experts $E_s$ and the gate $G$ of a *VFL_MoE* could be instantiated using different neural network architectures. For the sake of memory, communication band and computation saving, the following design choices are taken in our approach:

- Each expert $E_s$, for $s \in \{1, \ldots, m\}$, is simply implemented as a $|X_s|$-to-1 dimensional one-layer feed-forward net with linear activation functions (followed by a sigmoid transformation) —i.e., this net takes a numerical representation of a data instance in the feature space associated with the $s$-th data owner $DO_s$.
- The gate $G$ is implemented as a one-layer $|X_0|$-to-$m$ feed-forward network with linear activation functions (followed by a non-trainable aggregation module implementing the top-$k$-selection operator and the final Softmax transformation).

Based on the memory, time and energy/carbon footprint budget that has been settled in the VFL network, one can adopt more powerful neural architectures for both the gate and the experts. A viable way of pursuing this goal consists of providing each DO node with a sparse-MoE-based model, so obtaining a sort of hierarchical MoE classifier in the end. This allows for enhancing the representation and classification power of the federated classification model without incurring too high computing and energy costs by virtue of such a two-level scheme of conditional computation.

#### The proposed training method: algorithm `VFL_MoE`

Our approach to discovering a *VFL_MoE* classifier (of the form defined in Def. 2) consists of applying a novel federated training algorithm, referred to hereinafter as `VFL_MoE` algorithm.

Three hyper-parameters allow the user to flexibly control the computation and communication costs related to running the algorithm and applying the resulting *VFL_MoE* classifier on new data instances: the maximal number $e$ of training epochs (i.e., of distributed VFL rounds); a batch-reduction factor $r \in (0, 1]$ indicating the proportion of total instance batches utilized in the training process, in line with the *Repeated Random Sampling* approach as proposed in [33]; and the expert-selection factor model hyper-parameter $k$ determining how many experts' predictions the model will consider in classifying a data instance.

As further hyper-parameters, the algorithm includes the size $b$ of the mini-batches, the learning rates $\eta_g$ and $\eta_e$ employed in optimising the gates' and each expert's parameters, respectively.

**Algorithm 1** Pseudo-code of the distributed algorithm VFL_MoE.

**Data:** Tensors $X$ and $Y$ storing the input feature vectors $x_0^i, \ldots, x_m^i$ and class label $\hat{y}^i$ (for $i \in [1..N]$), respectively for all the data instances $d^i$ of a distributed dataset $D$ (cf. Def.1);
**Requires:** expert-selection factor $k$; max. number $e$ of epochs; batch-reduction factor $r$; batch size $b$; learning rates $\eta_g$ and $\eta_e$ for the gate's and experts' parameters, respectively.
**Result:** A *VFL_MoE* model $\langle \mathcal{N}_g, E_1, \ldots, E_m \rangle$ with optimized parameters $\Theta = [\Theta_g \Theta_1 \ldots \Theta_m]^t$.

1 The coordinator node $C$ and every data owner node $DO_s$ (with $s \in [1..m]$) concurrently initialize the parameters $\Theta_g$ and $\Theta_s$ of the gate sub-net $\mathcal{N}_g$ and of expert $E_s$, respectively;
2 $C$ generates a seed $\varepsilon \in \mathbb{N}$ to be used by all the nodes involved in the training for randomly sampling the same subset of data batches in each training epoch;
3 $C$ **sends** the values of the seed $\varepsilon$ and of the factor $r$ to all the DO nodes $DO_1, \ldots, DO_m$;
4 **foreach** epoch $e \in [1 \,..\, e]$ **do**
5 All the nodes sample a vector $\mathcal{I}$ of $\lceil r \cdot N \rceil$ data instance indices using the same seed $\varepsilon$;
6 Let $B_1, \ldots, B_{n_b}$, with $n_b = \lceil \frac{r \cdot N}{b} \rceil$, denote the training batches, regarded as sets of (data instance) indexes s.t. $B_i = \{\mathcal{I}(j') \mid j' \in [(i-1) \cdot b + 1 \,..\, i \cdot b]\}$ for any $i \in \{1, \ldots, n_b\}$;
7 **foreach** batch $B_i$ s.t. $i \in [1 \,..\, n_b]$ **do**
8 **foreach** data-instance index $j \in B_i$ **do in parallel**
9 Every node $DO_s$ performs a forward pass through its expert model $E_s$ on the current data instance $d^j$ (i.e. the one referred to by index $j$), to compute the respective logit $z_s^j = f_s(x_s^j; \Theta_s)$ and prediction $y_s^j = \sigma(z_s^j)$ (cf. Def. 2);
10 $C$ performs a forward pass through the gate $\mathcal{N}_g$ on the current data instance $d^j$, to compute the respective vector $w_g^j = g(x_0^j; \Theta_g)$ of expert weights ;
11 Every $DO_s$ node **sends** the logit and output values it has computed for the instances of the current batch to $C$, as an ordered list of pairs $(z_s^j, y_s^j)$ for all $j$ in $B_i$ ;
12 **foreach** data-instance index $j \in B_i$ **do**
13 $C$ computes the final prediction $\hat{y}^j = (w_g^j)^t [y_1^j \ldots y_m^j]^t$, the total per-instance loss $\mathcal{L}(x^j, \hat{y}^j; \Theta)$ (based on Eq. 3) and the partial derivative $\delta_s^j = \frac{\partial \mathcal{L}(x^j, \hat{y}^j; \Theta)}{\partial f_s(x_s^j)}$;
14 $C$ **sends** the derivatives $\{\delta_s^j \mid j \in B_i\}$ to each DO node $DO_s$ as an ordered list;
15 **do in parallel**
16 $C$ computes gradients $\{\nabla_{\Theta_g} \mathcal{L}(x^j, \hat{y}^j; \Theta) \mid j \in B_i\}$, aggregates them all into average one $\mathcal{G}_g$, and updates the parameters of gate $\mathcal{N}_g$ via $\Theta_g := \Theta_g - \eta_g \cdot \mathcal{G}_g$;
17 Every $DO_s$ computes gradients $\{\nabla_{\Theta_s} \mathcal{L}(x^j, \hat{y}^j; \Theta) \mid j \in B_i\}$, averages them all into $\mathcal{G}_s$ and updates the parameters of $E_s$ via $\Theta_s := \Theta_s - \eta_e \cdot \mathcal{G}_s$;
18 All DO nodes $DO_s$ apply their experts to the $N - \lceil r \cdot N \rceil$ instances that were not used in the last iteration of the main loop (Steps 4-17) and **send** the resulting logit and output values to $C$;
19 The parameters $\Theta_g$ of gate $\mathcal{N}_g$ are fine-tuned by making $C$ execute the loop over Steps 4-17 in isolation again –i.e. skipping Steps 9, 11, 14, 16 and 17– using the logit and output values it gathered in Step 18 and in Step 11 of the last iteration of the previous complete run of the loop;
20 **return** the updated version of *VFL_MoE* $\langle \mathcal{N}_g, E_1, \ldots, E_m \rangle$.

*Core computation steps* Conceptually, the proposed distributed training algorithm consists of three main phases (also reported in the flow diagram of Fig. 4):

1. First, in Step 1, a randomly-initialized *VFL_MoE* model is constructed and maintained by the different nodes of the VFL network, which include a coordinator $C$ and several data-owner $DO_1 \ldots, DO_m$, according to the model architecture specified in Def. 2. Moreover, prior to starting the very training process, the nodes set a common criterion for iterating across the data instances in a coordinated way by agreeing on which random seed they will employ to sample the same fraction $r$ of data instances at each training epoch (Steps 2–3).

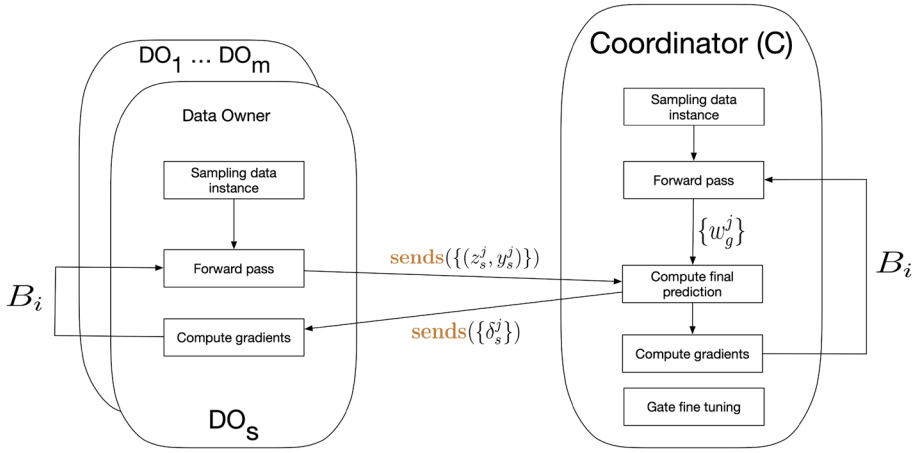


**Fig. 4** Flow diagram of our MoE-based VFL_MoE framework

2. The second phase consists of training the *VFL_MoE* model by executing a complete run of the main loop spanning over Steps 4 to 17. Conceptually, this loop encodes a standard mini-batch-based SGD-like procedure for optimizing the *VFL_MoE* model as a whole in an end-to-end way. More specifically, for each batch $B_i$ consisting of $b$ data instances: *(i)* first, in Steps 8–10, the gate and expert sub-models perform a forward pass on all the data instances associated with $B_i$ (specifically, every data owner node computes the respective logits and the predictions, while the coordinator computes the weights); *(ii)* then, in Steps 12–13, per-instance losses are computed (as functions of the model parameters $\Theta$), after assembling the sub-models' intermediate outputs $w_g^j, y_1^j, \ldots, y_m^j$ into an overall prediction $y^j$ for all $j$ in $B_i$; *(iii)* finally, in Steps 15–17, the per-instance gradients of each model parameter in $\Theta$ are computed via back-propagation,[6] averaged, and eventually used to update the parameters. A more detailed explanation of this phase is provided in the final part of this subsection, after the definition of the loss function (Def. 3).
3. The last core computation, summarized in Step 19, consists in fine-tuning the parameters $\Theta_g$ of the gate sub-net, while keeping the experts' parameters frozen. This is accomplished by making the sole coordinator node $C$ re-execute the training loop spanning over Steps 3–17, while skipping those steps entail novel calculations by the DO nodes, for the sake of efficiency.

*Communication steps* In addition to the core computation steps described above, the algorithm includes a number of communications between the coordinator node $C$ and the DO nodes. For the sake of presentation, these communications are denoted in the algorithm as occurrences of generic data-exchange action **send**. For the sake of simplicity and concreteness, we hereinafter assume all these communications are performed in a point-to-point fashion while noting that, in fact, some of them could be implemented

[6] For the sake of efficient computation, in the case of the experts' parameters in $\cup_s \Theta_s$, the back-propagation calculation was started by the coordinator node $C$ in Step 13 (with the computation of the partial derivatives $\{\delta_s^j \mid j \in B_i\}$); the procedure is then finalized in a per-expert local way in Step 17.

through more efficient collective communication patterns (e.g., using broadcast, scatter, gather, etc., operations when transferring data among the coordinator and the DO nodes). We pinpoint that most of the data-exchange operations are performed once for each optimization step (i.e., for each mini-batch of training instances) —let us abstract, for now, from the first communication operation performed in Step 3, during the initialization phase. Thus, the amount of data exchanged in each of these batch-wise communications is really small: just 2 and 1 scalars per batch instance from each $DO_s$ to $C$ (Step 11) and in the opposite direction (Step 14), respectively.

The fine-tuning procedure of Step 19 does not involve communications in itself, for it is carried out by the sole coordinator $C$ without any intervention from DO nodes. However, to enable this, $C$ must know, for all the instances in dataset $D$, the output (and logit) returned by the final version of the experts, obtained at the end of the training phase. To this purpose, in Step 18, $C$ collects from the other nodes the predictions of each associated expert for all the data instances that were not considered in the last training epoch.

*Loss function and details on the optimization steps* For the sake of technical completeness, let us illustrate the specific loss function that is employed in the training procedure to guide the optimization of the model parameters, namely $\Theta_g$ for the gate sub-net and $\Theta_s$ for all local expert $E_s$.

**Definition 3** Let $x = [x_0, \ldots, x_m]^t$ be the vectorial representation of any training data instance, where sub-vector $x_0$ and $x_1, \ldots, x_m$ store the values that $x$ takes on the shared features and the local private features of nodes $DO_1, \ldots, DO_m$. Let $\hat{y}$ be the ground-truth label of (the data instance represented by) $x$. The loss made by a *VFL_MoE* $\mathcal{N} = \langle \mathcal{N}_G, E_1, \ldots, E_m \rangle$, with $\mathcal{N}_G, E_1, \ldots, E_m$ parametrized by parameters $\Theta = [\Theta_g \Theta_1 \ldots, \Theta_m]^t$ is defined as follows:

$$\mathcal{L}(x, \hat{y}, \Theta) = -\frac{1}{m \cdot \sqrt{2\pi}} \log \left( 1 + \sum_{s=1}^{m} Q(x, \hat{y}, \Theta_s) \cdot g(x_0; \Theta_g)_s \right) \tag{3}$$

where $g(\cdot; \Theta_g)_s$ is the weight that the gate $\mathcal{N}_g$ is attributing, for the input data instance $x$, to the $s$-th expert prior to the top-$k$ transformation; $Q(x, \hat{y}, \Theta_s) = \exp(f_s(x_s; \Theta_s) \cdot (1 - 2\hat{y}))$ is a per-expert loss-like term that is meant to capture the prediction error of $E_s$,[7] and $f_s(x_s; \Theta_s)$ is the logit that the expert $E_s$ produces for $x$ (by only looking at its locally-known features) prior to the final application of a Sigmoid transformation —see Def. 2 for more details on the meaning of these terms. □

In a nutshell, to find a combination of parameters of the *VFL_MoE* model that (locally) minimizes this loss function, the optimization procedure performs the following major operations for each training instance $(x, \hat{y})$ (of each mini-batch considered in each epoch):

- In the forward pass, the predictions of all the experts are computed for $x$ in the DO nodes by applying each expert to the local representation of $x$ it has been provided with.

[7] Notably, the loss in Eq. 3, as discussed in both [50] and [51], has been proven to favour expert specialization.

- In the coordinator node $C$, all the expert predictions are combined with the output $g(x_0; \Theta_g)$ of the gate to obtain an overall class prediction and to compute, according to Eq. 3, the loss value $\mathscr{L}(x, \hat{y}; \Theta)$. The coordinator is also in charge of computing both the gradient $\nabla_{g(x_0;\Theta_g)} \mathscr{L}(x, \hat{y}, \Theta)$ of the output layer of the gate, and the partial derivatives $\frac{\partial \mathscr{L}(x,\hat{y},\Theta)}{\partial Q(x,\hat{y},\Theta_s)}$ for the logit nodes of all experts $E_s$.
- By back-propagating gradient $\nabla_{g(x_0;\Theta_g)} \mathscr{L}(x, \hat{y}; \Theta)$, $C$ can obtain the batch-wise aggregated gradient $\mathscr{G}_g = \texttt{avg}_{(x,\hat{y})} \left( \nabla_{\Theta_g} \mathscr{L}(x, \hat{y}; \Theta) \right)$ that it will exploit to greedily optimize the gate parameters.
- On the other hand, once provided with $\frac{\partial \mathscr{L}(x,\hat{y},\Theta)}{\partial Q(x,\hat{y},\Theta_s)}$, each $DO_s$ can derive (via back-propagation) the batch-wise average gradient $\mathscr{G}_g = \texttt{avg}_{(x,\hat{y})} \left( \nabla_{\Theta_g} \mathscr{L}(x, \hat{y}, \Theta) \right)$ eventually employed to optimize the parameters of expert $E_s$.

#### Analytical study of the algorithm's costs

*Computation costs* Any instance of the proposed federated classification model *VFL_MoE* consists of quite shallow and small sub-nets: a two-layer feed-forward gate $\mathscr{N}_g$ and $m$ expert models $E_1, \ldots, E_m$.

The gate sub-net, having an $m$-dimensional output, contains $m \cdot (d'_0 + 1) + d'_0 \cdot (d_0 + 1)$ parameters, where $d_0 = |X_0|$ and $d'_0$ is the number of neurons in its second layer. Each expert sub-net, having a one-dimensional output, contains instead $d_s + 1$ parameters where $d_s = |X_s|$, for a total of $m \cdot \sum_s (d_s + 1) = m \cdot d$ parameters in all the experts, where $d = |X|$ is the total number of (post-numericalization) input features plus $|X_0| \cdot (m - 1)$.

So assuming that less than $\max_{s=0}^{m} d_s < d \ll 10^4$ and that $m \ll 100$, as in the case study discussed in Sect. 5, each of these sub-nets consists of less than $4 \cdot 10^4$ floating-point numbers, so requiring quite a small amount of main/GPU memory to be stored and processed.

Let us now roughly estimate the total compute burden required by a complete execution of the proposed federated training method `VFL_MoE`, illustrated in Algorithm 1, and described in Section 4.2.

For the sake of simplicity and generality, let us here focus on the total number of floating-point operations (FLOPs) performed in this process as a proxy for its total energy cost. This metric, reflecting the number of elementary arithmetic (e.g., multiplication, addition, division, subtraction and transcendental) operations performed in the computation, will let us characterize the efficiency of the proposed algorithm independently of the hardware and software infrastructure over which it is run, so allowing for broader comparability with existing and future solutions.

Considering the simple feed-forward architecture of the gate and experts' sub-net, it is easily seen that a forward pass through each layer of them with $d_{IN}$ neurons and $d_{OUT}$ neurons requires $d_{OUT} \cdot (d_{IN} + 2)$ FLOPs —the latter value is here to account for bias addition and non-linearity computations. This means that $m \cdot (d'_0 + d_i + 4) + d'_0 \cdot (d_0 + 2)$ FLOPs per forward step are performed when training the model as a whole, while the fine-tuning of the gate requires

$m \cdot (d'_0 + 2) + d'_0 \cdot (d_0 + 2)$, for a total of $C_{for} = m \cdot (2d'_0 + d_i + 6) + 4d'_0 \cdot (d_0 + 2)$. Approximating the cost of each back-propagation plus gradient-related computation steps with $2 \cdot C_{for}$, and considering that the total number of training steps performed is equal to $e \cdot r \cdot N$ (with $N$ denoting the number of instances in the training dataset $D$), we come at the following overall estimate for the cost of training algorithm (Algorithm 1): $3 \cdot e \cdot r \cdot N \cdot m \cdot (2d'_0 + d_i + 6) + 4d'_0 \cdot (d_0 + 2) = \Theta(e \cdot r \cdot N \cdot P)$, where $P$ is the total number of parameters in the model.

Note that, at test time, a *VFL_MoE* discovered with the algorithm can make a class prediction in a speedy and compute-efficient way by simply performing a forward pass throughout the gate and the $k$ experts that have been chosen for the prediction: this computation just requires $(d'_0 + d_i + 4) + d'_0 \cdot (d_0 + 2)$ FLOPs per test instance.

*Communication costs* The total number of communications performed in the main loop of Algorithm 1 (assuming that all messages are exchanged through pairwise point-to-point communications) is equal to $e \cdot m \cdot \lfloor \frac{r \cdot N}{b} \rfloor$. The maximum amount of data exchanged in each of these communications just corresponds to $2 \cdot b$ floating-point numbers. In addition, $m$ communications are performed in both Step 2 and Step 18, involving the transmission of 1 and $2 \cdot (N - \lceil r \cdot N \rceil)$ floating-point numbers, respectively.

Thus, in a complete run of the algorithm, the total number of communications performed is $2 \cdot m + e \cdot m \cdot \lfloor \frac{r \cdot N}{b} \rfloor$, and the total number of floating-point numbers exchanged is $\Theta(e \cdot m \cdot r \cdot N)$, assuming that $e \cdot r \geq 1$.

Please notice that, larger amounts of data are exchanged instead in traditional FL approaches requiring the sharing of higher dimensional gradients/parameters at each (per mini-batch) optimization step, as well as in the approach proposed in [30], which also require transferring embedded versions of all the local raw data from the data owners to the coordinator.

*Final remarks* In conclusion, under the mild assumption $e \cdot r \geq 1$, the computation and communication costs of the proposed training algorithm `VFL_MoE` (reported in Algorithm 1) are linear in the total number $\lceil r \cdot N \rceil$ of instances processed across all the training epochs. This allows for easily controlling these costs by acting on hyper-parameter $r$.

## Experimental results

This section aims to evaluate the capability of the proposed algorithm `VFL_MoE` in accurately detecting malicious behaviors using the *KronoDroid* dataset [52], a recently-compiled benchmark dataset featuring an extensive collection of malware samples affecting Android OS-based systems from 2008 to 2020. To study the performance of `VFL_MoE` in a different application scenario, we also extended our experimental assessment to the publicly-available *Adult* dataset [53].

### Testbed

#### *Datasets*

The KronoDroid dataset is a comprehensive collection of Android application samples, both benign and malicious, spanning from 2008 to 2020. This dataset is characterized by its timestamped data samples, covering a wide time range. Each sample is described using a set of 489 features, 289 of which are dynamic and the others static.

This dataset is widely adopted as a benchmark in cybersecurity domains, particularly for studying the evolution of Android malware and the development of detection mechanisms.

*KronoDroid* includes 41,382 malware samples across 240 malware families and 36,755 benign applications. It is the most extensive dataset with hybrid features focused on Android platforms. The dataset is divided into two distinct sub-datasets based on emulators and real devices. This division facilitates analysis across different execution environments. This categorization enabled the effective vertical partitioning of the dataset into four segments, namely `System Calls`, `Permissions`, `Intents`, and `Others`, which proved instrumental in testing our vertical federated learning framework. These groups encompass 289, 173, 7, and 8 attributes, respectively. The dataset was normalized, and all the features exhibiting a strong correlation with the label `Malware` (i.e., `Detection_Ratio`) or deemed non-contributory (such as `sha256`, `EarliestModDate`, `HighestModDate`, and `MalFamily`) were removed.

The *Adult* dataset encompasses information extracted from census forms about various households. It comprises 14 distinct attributes, each capturing different socioeconomic factors. The objective of this dataset is to predict whether a given household possesses an income surpassing the $50,000 threshold. The original Adult dataset includes 14 features, delineated into six continuous and eight categorical attributes. Notably, the continuous features are discretized into quantiles, with each quantile subsequently represented by a binary feature. Similarly, categorical features, characterized by $m$ distinct categories, are transformed into $m$ binary features.

#### *Experimental setup*

*Dataset preparation* The datasets in our analysis were divided into three subsets: training, validation, and test. We allocated 80% of the dataset, randomly chosen, for training. The remaining 20% was set aside for testing. Additionally, 20% of the training subset was used as a validation set.

In the framework of our experiments, we defined a subset of dataset features as public. Specifically, for the *KronoDroid* dataset, this subset includes all features from the `Intents` group and a carefully selected set of 5 features from the `Permissions` group, namely `WRITE_EXTERNAL_STORAGE`, `RECEIVE_BOOT_COMPLETED`, `RECEIVE_SMS`, `READ_SMS`, and `GET_TASKS`. These public features are shared across all local nodes as well as the coordinating node, as illustrated in Fig. 3. The rest of the features are retained as private, with each local node securely holding its respective subset. This configuration ensures that while certain non-critical features are made available for collective learning, sensitive or node-specific data remains confined locally, adhering to the principles of vertical federated learning.

Regarding the *Adult* dataset, we randomly distributed the features into four distinct parts. The first part consists of common attributes the gate uses to determine the allocation of instances to various experts. The other three parts are distributed equally among three distinct organizations, ensuring a balanced data division for collaborative yet private learning.

*Model variants* To more thoroughly assess the strengths and weaknesses of `VFL_MoE` in terms of accuracy, privacy preservation, and communication costs, we examined a centralized variant of it, namely `Centr_MoE`. This model deviates from `VFL_MoE` in its commitment to privacy and computational and communication requirements. It represents a distinct approach within the crucial spectrum of accuracy, privacy, and communication costs in a VFL setting.

Specifically, `Centr_MoE` is conceptualized as a theoretical upper bound in our analysis. It embodies a fully centralized model where the MoE is localized in the Coordinator node. This configuration, not adhering to the typical privacy standards of any VFL setting, makes `Centr_MoE` an ideal yet not realizable variant. It utilizes unmasked and unencrypted raw data, including both common and local node-specific features —both MoE's gate and experts in the coordinator utilize the entire set of 477 features. Although transferring such raw data to the central node may entail non-negligible communication costs, this occurs only once at the onset of the training phase. This contrasts `VFL_MoE`, which requires data transfer after every training batch. While `Centr_MoE` entirely overlooks privacy considerations, potentially leading to privacy risks, its unrestricted access to complete information positions it as an idealized benchmark for effectiveness performances.

*Parameters* In the experimental evaluation of our approach `VFL_MoE`, we employed a two-tiered strategy for parameter configuration. This involved setting some hyper-parameters based on empirical findings while varying two critical parameters, $k$ and $r$, to analyze and assess the behaviour of our approach.

In particular, $k$ is a crucial hyper-parameter representing the number of experts the MoE model utilizes, combining them to predict the class label. Its value was varied from 1 to 3. This variation aims to understand how the number of experts influences the model's performance and robustness.

Hyper-parameter $r$ serves the purpose of controlling the number of batches (and related optimization steps) performed per training epochs, according to the *Repeated Random Sampling* method proposed in [33]. In other words, $r$ represents a data/compute reduction factor that is meant to be applied in the model training process. We made $r$ vary from 0.075 to 1, with incremental steps of 0.125. This range corresponds to using 7.5% up to 100% of all available batches, respectively. This variation is intended to evaluate the impact of the total number of (batch-wise) optimization steps on the method performances.

Fixed hyper-parameters were established as follows: the maximum number of training epochs was set at $e = 40$. Learning rates were set as follows: $\eta_e = 10^{-4}$ for the experts and $\eta_g = 10^{-3}$ for the gate. Additionally, we limited the number of experts per organization (where applicable) to a maximum of one and standardized the batch size across experiments at $b = 64$. Each expert consists of a single linear layer, while the coordinator model comprises two linear layers internally configured with 512 neurons each. The coordinator model also includes dual-head linear layers for computing the two outputs for the classification task, with the first primarily used for debugging and the second containing the weights assigned to the experts. All the experiments in the following subsections have been averaged over 30 runs.

*Effectiveness metrics* In this study, we address the problem at hand as a binary classification task, distinguishing between two primary classes: the 'normality' class, representing non-malicious entities, and the 'attack' class, indicative of malware presence —this binary framework is crucial for a targeted analysis in distinguishing malicious from benign instances. The metrics are computed explicitly in relation to the prediction of the malware class, which is of interest in our context.

Specifically, different metrics are adopted in cybersecurity to evaluate various aspects of a malware detection model's performance. Each metric, with its own strengths, collectively provides a comprehensive picture of the model's effectiveness. Accurately detecting malware while minimizing false alarms is crucial for maintaining system integrity and user trust. In this context, we discuss four key metrics: *Accuracy Score*, *Area Under the Receiver Operating Characteristics*, *F1-score*, and *False Positive Rate*.

- *Accuracy Score* (`ACC`) is calculated as:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

  It measures the proportion of total correct predictions (both malware and non-malware) made by the model out of all predictions. A high accuracy score indicates a generally reliable model across various scenarios in malware detection. However, it is essential to note that in cases of imbalanced datasets-where one class (e.g., malware) is much rarer than the other-accuracy alone might be misleading.
- *Area Under the ROC* (`AUC`) is derived by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at different thresholds and calculating the area under this curve. This metric is critical, especially in scenarios with imbalanced classes, as it measures the model's ability to discriminate between classes (malware and benign) at various thresholds. A model with a high `auroc` is apt at distinguishing malware from non-malware, making it valuable for ensuring that genuine threats are not missed.
- *F1 Score* (`F1`) is defined as:

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

  This metric is particularly relevant when there is a need to balance between Precision and Recall. In malware detection, this means correctly identifying actual malware (Recall) while minimizing the misclassification of benign software as malware (Precision). It is beneficial when false positives and false negatives have serious implications.
- *False Positive Rate* (`FPR`) is given by:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

  It is a critical metric in environments where the cost of false alarms is high. In cybersecurity, a high FPR could mean unnecessary resource allocation for investigating

benign instances flagged as malicious, potentially leading to distrust in the detection system.

Each of these metrics provides specific insights into a model's performance. However, they should not be considered in isolation. A comprehensive evaluation of a malware detection model necessitates considering the trade-offs and specific contexts in which the model operates. Balancing these metrics effectively ensures the development of a robust, reliable, and efficient malware detection system.

*Competitor and Baseline approaches* In this study, we sought a comprehensive understanding of how our VFL approach, denoted as `VFL_MoE`, compares within the landscape of accuracy maximization, privacy preservation, and communication and computational demands minimization. To this end, we considered two key benchmarks: `SVFL` and `Baseline`.

`SVFL` [30] is the only known competitor in the literature that integrates an MoE into a VFL setting for classification purposes (a detailed description of the differences between `VFL_MoE` and `SVFL` is provided in Sect. 6). Unlike `VFL_MoE`, which allows the combination of different experts' predictions (parameter $k$) for the task of classification, `SVFL` is limited to using only the single best-performing expert ($k = 1$) during inference. This distinction could represent an essential aspect in evaluating their relative accuracy performances. In addition, while `SVFL` may benefit from using a more extensive data set in each training cycle ($r = 1.0$) than `VFL_MoE` where $r \leq 1.0$, this comes at the expense of increased communication and computational requirements. Regarding privacy, both approaches adhere to VFL's foundational requirements, such as data segregation/siloing. However, `VFL_MoE` exhibits a more robust privacy profile by avoiding transferring embedded data to the coordinator node, a requirement in `SVFL` for the MoE's gate to select appropriate experts. This aspect not only enhances privacy by preventing potential data exposure but also sensibly reduces communication costs in `VFL_MoE`.

When directly compared under the same setting with $k = 1$, `VFL_MoE` both reduces the risk of private information leakage for the data owners, and it is less demanding than `SVFL` in terms of communication and computation costs. The comparison regarding potential accuracy achievements is more nuanced. Indeed, when setting hyperparameter $r \leq 1.0$, `VFL_MoE` is allowed to exploit a smaller number of training instances than `SVFL`, with an increased risk of missing some proper supervision signal. However, in application scenarios where high accuracy levels are required, one can allow `VFL_MoE` to compensate for this by combining the predictions of more experts at inference time.

`Baseline`, on the other hand, serves a distinct role in our evaluation. It represents an extreme use case where each local model, restricted to data from a single participating organization, operates independently without any MoE mechanism. As such, the parameter $k$ is inapplicable in this context. Specifically, `Baseline` aims to simulate a sort of "extreme scenario" with maximal privacy (by keeping data localized) and minimal communication costs (no data transfer to the Coordinator) but without any inter-node cooperation. This approach processes both public and non-public features as `VFL_MoE` does, but uniquely within the confines of each participating organization, resulting in three distinct models. These models are autonomous, and their accuracy metrics are averaged to form a consolidated output.

Based on its characteristics, `Baseline` offers a unique perspective, serving as a benchmark for our VFL-based approach `VFL_MoE` by demonstrating the effectiveness (or lack thereof) of using isolated local models for classification. Indeed, if `Baseline` would perform adequately, it would suggest that local models alone could suffice for classification tasks, negating the need for more complex federated structures. In addition, the absence of communication costs in `Baseline` negates the relevance of the parameter $r$ for reducing data batches during training, setting it at $r = 1.0$ by default. Notably, `Baseline` does not align with the VFL requirements due to its lack of model combination or cooperation, and thus, in this sense, it is not a direct competitor but rather a lower-end reference point.

In summary, comparing `VFL_MoE` with `SVFL` and `Baseline` across the spectra of accuracy, privacy, and resource efficiency provides valuable insights into the strengths and limitations of these approaches in a VFL context. While `VFL_MoE` shows promise in balancing these aspects, the unique characteristics of `SVFL` and `Baseline` offer essential benchmarks for evaluating the relevance of our proposed method.

## Test results

### *Analysis of performance on the KronoDroid dataset*

In Table 1, we delve into a detailed comparison of our VFL approach, `VFL_MoE`, with its ideal upper bound counterpart, `Centr_MoE`, across varying values of parameters $k \in \{1, 2, 3\}$ and $r \in \{0.075, \ldots, 1.0\}$.

A first observation descending from outcomes in Table 1 is the substantial robustness of `Centr_MoE` against variations in both $k$ and $r$. This robustness can be presumably attributed to the comprehensive availability of (both public and non-public) features for the MoE's gate in `Centr_MoE`. This extensive data access possibly allows the gate to select the most suitable experts more precisely, thereby reducing the need to aggregate multiple predictions to compensate for potential inaccuracies in the expert selection. Further supporting this, we can also observe in Fig. 5 that `Centr_MoE` tends to flatten its performance curve quite early, already around $r = 0.375$. This early flattening suggests that with access to all unmasked data, the MoE's gate in `Centr_MoE` can more effectively identify patterns in the input data, thereby enhancing its expert selection with lesser training data. However, it is again crucial to acknowledge that `Centr_MoE`, by its very design, remains an unattainable ideal model in practical VFL scenarios. Thus, it serves primarily as a theoretical upper bound for comparison with our federated approach `VFL_MoE`.

Examining the behaviour of `VFL_MoE` in Table 1 (and also in Table 2), it is interesting to note that its performance is not dramatically inferior to that of the ideal model `Centr_MoE` in terms of most accuracy metrics for all values of $k$ and $r$ —only for *FPR* this difference is more pronounced. The gap between `VFL_MoE` and `Centr_MoE` continues to narrow with the increase of these parameters. For instance, with $k = 1$ and $r = 0.25$, `VFL_MoE` shows a performance gap from `Centr_MoE` in terms of AUC, ACC, FPR, and F1 of $-2.1\%$, $-3.2\%$, $-83.7\%$, and $-3.2\%$, respectively. When we increase $r$ to 0.75, this gap reduces to $-1.8\%$, $-2.7\%$, $-68.1\%$, and $-2.4\%$ for the same metrics. As expected, this trend of shrinking gaps continues with higher $k$ values. For $k = 2$, the performance difference further decreases, with $-1.1\%$, $-2.1\%$, $-41.5\%$, and $-2.1\%$ when

**Table 1** Detailed performance evaluation of the VFL-based approach `VFL_MoE` and its upper bound variant `Centr_MoE` on the *KronoDroid* dataset

| *k* | *Approach* | *r* | *AUC* (↑) | *ACC* (↑) | *FPR* (↓) | *F1* (↑) |
|---|---|---|---|---|---|---|
| 1 | `VFL_MoE` | 0.075 | 0.927 ± 0.003 | 0.857 ± 0.009 | 0.088 ± 0.013 | 0.857 ± 0.012 |
| | | 0.125 | 0.943 ± 0.002 | 0.882 ± 0.003 | 0.082 ± 0.011 | 0.884 ± 0.003 |
| | | 0.250 | 0.956 ± 0.001 | 0.904 ± 0.001 | 0.079 ± 0.005 | 0.907 ± 0.001 |
| | | 0.375 | 0.959 ± 0.001 | 0.908 ± 0.002 | 0.082 ± 0.003 | 0.912 ± 0.002 |
| | | 0.500 | 0.961 ± 0.001 | 0.914 ± 0.001 | 0.080 ± 0.002 | 0.917 ± 0.001 |
| | | 0.625 | 0.963 ± 0.003 | 0.916 ± 0.003 | 0.078 ± 0.004 | 0.920 ± 0.003 |
| | | 0.750 | 0.963 ± 0.002 | 0.915 ± 0.003 | 0.079 ± 0.005 | 0.919 ± 0.003 |
| | | 0.875 | 0.964 ± 0.001 | 0.916 ± 0.002 | 0.079 ± 0.003 | 0.920 ± 0.002 |
| | | 1.000 | 0.964 ± 0.002 | 0.916 ± 0.003 | 0.081 ± 0.003 | 0.920 ± 0.003 |
| | `Centr_MoE` | 0.075 | 0.962 ± 0.002 | 0.896 ± 0.003 | 0.041 ± 0.002 | 0.895 ± 0.004 |
| | | 0.125 | 0.972 ± 0.001 | 0.924 ± 0.002 | 0.040 ± 0.003 | 0.925 ± 0.002 |
| | | 0.250 | 0.977 ± 0.001 | 0.934 ± 0.001 | 0.043 ± 0.003 | 0.937 ± 0.001 |
| | | 0.375 | 0.980 ± 0.001 | 0.937 ± 0.001 | 0.045 ± 0.002 | 0.939 ± 0.002 |
| | | 0.500 | 0.980 ± 0.001 | 0.938 ± 0.001 | 0.047 ± 0.001 | 0.941 ± 0.002 |
| | | 0.625 | 0.981 ± 0.001 | 0.941 ± 0.002 | 0.045 ± 0.002 | 0.943 ± 0.002 |
| | | 0.750 | 0.981 ± 0.001 | 0.940 ± 0.002 | 0.047 ± 0.002 | 0.942 ± 0.002 |
| | | 0.875 | 0.981 ± 0.001 | 0.940 ± 0.002 | 0.048 ± 0.002 | 0.942 ± 0.002 |
| | | 1.000 | 0.981 ± 0.001 | 0.941 ± 0.002 | 0.046 ± 0.003 | 0.943 ± 0.002 |
| 2 | `VFL_MoE` | 0.075 | 0.953 ± 0.002 | 0.880 ± 0.003 | 0.061 ± 0.004 | 0.879 ± 0.002 |
| | | 0.125 | 0.958 ± 0.004 | 0.901 ± 0.003 | 0.059 ± 0.005 | 0.903 ± 0.003 |
| | | 0.250 | 0.967 ± 0.002 | 0.915 ± 0.003 | 0.058 ± 0.004 | 0.917 ± 0.003 |
| | | 0.375 | 0.970 ± 0.001 | 0.919 ± 0.004 | 0.062 ± 0.004 | 0.922 ± 0.004 |
| | | 0.500 | 0.972 ± 0.001 | 0.922 ± 0.002 | 0.064 ± 0.002 | 0.925 ± 0.002 |
| | | 0.625 | 0.973 ± 0.002 | 0.925 ± 0.003 | 0.062 ± 0.003 | 0.928 ± 0.003 |
| | | 0.750 | 0.973 ± 0.002 | 0.925 ± 0.002 | 0.063 ± 0.004 | 0.928 ± 0.002 |
| | | 0.875 | 0.974 ± 0.001 | 0.926 ± 0.002 | 0.063 ± 0.004 | 0.929 ± 0.002 |
| | | 1.000 | 0.975 ± 0.001 | 0.926 ± 0.002 | 0.064 ± 0.003 | 0.929 ± 0.002 |
| | `Centr_MoE` | 0.075 | 0.966 ± 0.002 | 0.900 ± 0.003 | 0.037 ± 0.002 | 0.899 ± 0.003 |
| | | 0.125 | 0.973 ± 0.001 | 0.924 ± 0.002 | 0.038 ± 0.002 | 0.926 ± 0.002 |
| | | 0.250 | 0.978 ± 0.001 | 0.935 ± 0.001 | 0.041 ± 0.002 | 0.937 ± 0.001 |
| | | 0.375 | 0.980 ± 0.001 | 0.937 ± 0.002 | 0.044 ± 0.002 | 0.939 ± 0.002 |
| | | 0.500 | 0.980 ± 0.001 | 0.938 ± 0.001 | 0.046 ± 0.002 | 0.941 ± 0.001 |
| | | 0.625 | 0.981 ± 0.001 | 0.941 ± 0.002 | 0.044 ± 0.002 | 0.943 ± 0.002 |
| | | 0.750 | 0.981 ± 0.001 | 0.940 ± 0.002 | 0.047 ± 0.002 | 0.942 ± 0.002 |
| | | 0.875 | 0.981 ± 0.001 | 0.940 ± 0.002 | 0.048 ± 0.002 | 0.942 ± 0.002 |
| | | 1.000 | 0.981 ± 0.001 | 0.941 ± 0.002 | 0.046 ± 0.003 | 0.943 ± 0.002 |

**Table 1** (continued)

| *k* | *Approach* | *r* | *AUC* (↑) | *ACC* (↑) | *FPR* (↓) | *F1* (↑) |
|---|---|---|---|---|---|---|
| 3 | VFL_MoE | 0.075 | 0.950 ± 0.002 | 0.879 ± 0.002 | 0.065 ± 0.003 | 0.878 ± 0.002 |
| | | 0.125 | 0.958 ± 0.002 | 0.895 ± 0.002 | 0.061 ± 0.003 | 0.896 ± 0.002 |
| | | 0.250 | 0.965 ± 0.001 | 0.910 ± 0.002 | 0.062 ± 0.004 | 0.912 ± 0.002 |
| | | 0.375 | 0.967 ± 0.001 | 0.913 ± 0.003 | 0.066 ± 0.003 | 0.916 ± 0.003 |
| | | 0.500 | 0.969 ± 0.001 | 0.915 ± 0.002 | 0.067 ± 0.001 | 0.918 ± 0.002 |
| | | 0.625 | 0.970 ± 0.001 | 0.919 ± 0.003 | 0.067 ± 0.003 | 0.922 ± 0.003 |
| | | 0.750 | 0.970 ± 0.001 | 0.921 ± 0.003 | 0.066 ± 0.003 | 0.924 ± 0.003 |
| | | 0.875 | 0.971 ± 0.001 | 0.922 ± 0.002 | 0.066 ± 0.004 | 0.925 ± 0.002 |
| | | 1.000 | 0.972 ± 0.001 | 0.922 ± 0.002 | 0.068 ± 0.003 | 0.925 ± 0.002 |
| | Centr_MoE | 0.075 | 0.967 ± 0.001 | 0.902 ± 0.002 | 0.036 ± 0.003 | 0.901 ± 0.003 |
| | | 0.125 | 0.974 ± 0.001 | 0.925 ± 0.002 | 0.038 ± 0.002 | 0.926 ± 0.002 |
| | | 0.250 | 0.978 ± 0.001 | 0.935 ± 0.001 | 0.041 ± 0.001 | 0.937 ± 0.001 |
| | | 0.375 | 0.980 ± 0.001 | 0.937 ± 0.002 | 0.045 ± 0.002 | 0.939 ± 0.002 |
| | | 0.500 | 0.980 ± 0.001 | 0.938 ± 0.001 | 0.046 ± 0.002 | 0.941 ± 0.001 |
| | | 0.625 | 0.981 ± 0.001 | 0.941 ± 0.002 | 0.044 ± 0.002 | 0.943 ± 0.002 |
| | | 0.750 | 0.981 ± 0.001 | 0.940 ± 0.002 | 0.047 ± 0.002 | 0.942 ± 0.002 |
| | | 0.875 | 0.981 ± 0.001 | 0.940 ± 0.002 | 0.048 ± 0.002 | 0.942 ± 0.002 |
| | | 1.000 | 0.981 ± 0.001 | 0.941 ± 0.002 | 0.046 ± 0.003 | 0.943 ± 0.002 |

This table presents a comprehensive analysis of the performance metrics at varying configurations of the number of expert predictions combined at inference time (parameter $k \in \{1, 2, 3\}$) and the batch-reduction factor for the number of training batch iterations $r \in \{0.075, \ldots, 1.0\}$

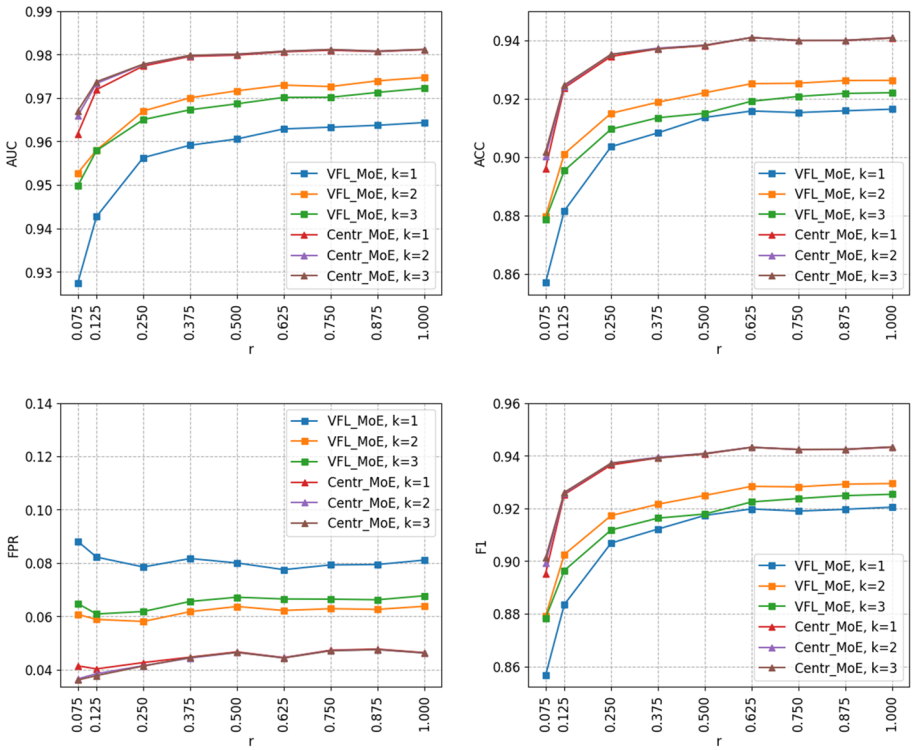


**Fig. 5** Comparative analysis on the *KronoDroid* dataset of different configurations of VFL_MoE and its ideal upper-bound variant, Centr_MoE, across a range of values for the parameter $k \in \{1, 2, 3\}$. The comparison takes into account various levels of the batch-reduction factor $r \in \{0.075, \ldots, 1\}$ during the training phase

**Table 2** Comparative analysis on the *KronoDroid* dataset of different models: the proposed method VFL_MoE, its ideal upper-bound variant Centr_MoE, Baseline, and competitor SVFL

| *k* | *Approach* | *r* | *AUC* (↑) | *ACC* (↑) | *FPR* (↓) | *F1* (↑) |
|---|---|---|---|---|---|---|
| 1 | Centr_MoE | 0.25 | 0.977 | 0.934 | 0.043 | 0.937 |
| | VFL_MoE | | 0.956 | 0.904 | 0.079 | 0.907 |
| | Centr_MoE | 0.50 | 0.980 | 0.938 | 0.047 | 0.941 |
| | VFL_MoE | | 0.961 | 0.914 | 0.080 | 0.917 |
| | Centr_MoE | 0.75 | 0.981 | 0.940 | 0.047 | 0.942 |
| | VFL_MoE | | 0.963 | 0.915 | 0.079 | 0.919 |
| 2 | Centr_MoE | 0.25 | 0.978 | 0.935 | 0.041 | 0.937 |
| | VFL_MoE | | 0.967 | 0.915 | 0.058 | 0.917 |
| | Centr_MoE | 0.50 | 0.980 | 0.938 | 0.046 | 0.941 |
| | VFL_MoE | | 0.972 | 0.922 | 0.064 | 0.925 |
| | Centr_MoE | 0.75 | 0.981 | 0.940 | 0.047 | 0.942 |
| | VFL_MoE | | 0.973 | 0.925 | 0.063 | 0.928 |
| 1 | SVFL [30] | 1.0 | 0.962 | 0.915 | 0.077 | 0.919 |
| - | Baseline | 1.0 | 0.940 | 0.867 | 0.110 | 0.870 |

All these methods were tested with expert-selection factor, $k \in \{1, 2\}$, except for Baseline (which lacks this hyper-parameter). Performance metrics are also reported for Centr_MoE and VFL_MoE across intermediate values of the batch-reduction factor hyper-parameter $r \in \{0.25, 0.50, 0.75\}$. Note that SVFL and Baseline do not employ any method to reduce the number of training batches hence $r = 1.0$ for both

$r = 0.25$, and $-0.8\%$, $-1.6\%$, $-34.0\%$, and $-1.5\%$ when $r = 0.75$ for AUC, ACC, FPR, and F1, respectively.

However, augmenting $k$ and $r$ to narrow performance gaps is not indefinitely beneficial. Indeed, a point of diminishing returns is reached, beyond which further increases in these parameters do not translate into commensurate performance enhancements. In our specific case, this appears to happen when moving from $k = 2$ to $k = 3$. In this scenario, increasing the value of $r$ and thereby trading off a substantial worsening in efficiency costs does not justify the marginal improvements in performance metrics. This behaviour, visually represented in Fig. 5, suggests that a configuration with $k = 2$ and $r = 0.25$ probably offers the most beneficial balance between performance improvements and efficiency costs in our context.

Table 2 allows for comparing the proposed VFL_MoE method with SVFL [30], the only existing approach to VFL that uses a MoE-based architecture, as well as with the Baseline method introduced in Sect. 5.1. It is important to note that even when compared in the same setting with $k = 1$, VFL_MoE begins to perform nearly on par with SVFL already when approximately more than half (i.e., $r \geq 0.5$) of the training batches used for SVFL are employed for VFL_MoE as well. Only in the more stringent case, where only 25% of the training batches used for SVFL are utilized for VFL_MoE, our model performs slightly worse, with marginal worsening in AUC, ACC, FPR, and F1 by $-0.6\%$, $-1.2\%$, $-2.6\%$, and $-1.3\%$, respectively, while benefiting from just a quarter of the computation cost for training. This further proves the robustness of VFL_MoE to (even drastic) reductions in training data availability. This shows that, even when selecting only one expert to classify each test instance ($k = 1$), as done by SVFL, VFL_MoE can outperform SVFL, provided that the fraction of data sampled per epoch is not so small. The incapability of SVFL to fully exploit the advantage of using the training dataset as a whole can be explained by the fact that its gate is

**Table 3** Comparative analysis on the *Adult* dataset of different models: the proposed approach VFL_MoE, its ideal upper-bound variant Centr_MoE, Baseline, and competitor SVFL

| *k* | *Approach* | *r* | *AUC* (↑) | *ACC* (↑) | *FPR* (↓) | *F1* (↑) |
|---|---|---|---|---|---|---|
| 1 | Centr_MoE | 0.25 | 0.896 | 0.816 | 0.175 | 0.666 |
| | VFL_MoE | | 0.874 | 0.810 | 0.159 | 0.639 |
| | Centr_MoE | 0.50 | 0.899 | 0.840 | 0.114 | 0.674 |
| | VFL_MoE | | 0.890 | 0.840 | 0.094 | 0.640 |
| | Centr_MoE | 0.75 | 0.900 | 0.847 | 0.086 | 0.675 |
| | VFL_MoE | | 0.893 | 0.842 | 0.077 | 0.654 |
| 2 | Centr_MoE | 0.25 | 0.897 | 0.815 | 0.179 | 0.666 |
| | VFL_MoE | | 0.887 | 0.820 | 0.151 | 0.639 |
| | Centr_MoE | 0.50 | 0.899 | 0.840 | 0.114 | 0.674 |
| | VFL_MoE | | 0.894 | 0.842 | 0.090 | 0.657 |
| | Centr_MoE | 0.75 | 0.900 | 0.847 | 0.086 | 0.675 |
| | VFL_MoE | | 0.894 | 0.844 | 0.071 | 0.657 |
| 1 | SVFL [30] | 1.0 | 0.877 | 0.830 | 0.081 | 0.604 |
| – | Baseline | 1.0 | 0.854 | 0.815 | 0.095 | 0.582 |

All the methods but Baseline were tested with expert-selection factor, $k \in \{1, 2\}$. For the sake of readability, performance results are reported for Centr_MoE and VFL_MoE across intermediate values of the batch-reduction factor $r \in \{0.25, 0.50, 0.75\}$. Note that SVFL and Baseline do not employ any method to reduce the number of training batches, hence $r = 1.0$ for both

provided with a complete but embedded (for the sake of privacy preservation) view of the data instances, which may not suffice for discovering an effective expert selection function.

Following this trend, when using $k = 2$, already for the case $r = 0.25$, the performances of VFL_MoE are comparable or even better (at least in terms of AUC and FPR) than those of SVFL and become significantly better when $r \geq 0.5$. These trends also show slight improvements for the case of $k = 3$, although this is not explicitly detailed in Table 2 but can still be observed from the detailed outcomes in Table 1.

Regarding the comparison with Baseline, VFL_MoE demonstrates an unequivocal advantage, regardless of the configuration of parameters $k$ and $r$. As the figures in Table 2 suggest, VFL_MoE consistently outperforms Baseline in all scenarios. This supports an essential aspect of our study: using isolated local models for classification, as Baseline does, is inadequate for achieving suitable performance in our context. This underscores the need for more sophisticated federated approaches like VFL_MoE, ensuring cooperation/combination of different views on the data provided by models computed in distributed local nodes.

#### *Analysis of performance on the adult dataset*

The evaluation of VFL_MoE's performance on the *Adult* dataset employs the same methodology used for the *KronoDroid* dataset (see Table 2). Results are detailed in Table 3, offering a comprehensive comparison with other models and showcasing VFL_MoE's performance across different datasets.

The analysis of VFL_MoE on the *Adult* dataset, as shown in Table 3, reveals trends similar to those observed in the *KronoDroid* dataset. Specifically, VFL_MoE closely approaches the performance of the ideal model Centr_MoE in most accuracy metrics

**Table 4** Ablation study for the KronoDroid dataset

| *Approach* | *AUC* (↑) | *ACC* (↑) | *FPR* (↓) | *F1* (↑) |
|---|---|---|---|---|
| VFL_MoE (*k*=2) | 0.975 | 0.926 | 0.064 | 0.929 |
| VFL_MoE (*k*=1) | 0.964 | 0.916 | 0.081 | 0.920 |
| Random Ensemble (*k*=2) | 0.961 | 0.902 | 0.086 | 0.905 |
| Random Ensemble (*k*=1) | 0.942 | 0.868 | 0.110 | 0.872 |

**Table 5** Ablation study for Adult dataset

| *Approach* | *AUC* (↑) | *ACC* (↑) | *FPR* (↓) | *F1* (↑) |
|---|---|---|---|---|
| VFL_MoE (*k*=2) | 0.896 | 0.845 | 0.066 | 0.659 |
| VFL_MoE (*k*=1) | 0.895 | 0.844 | 0.072 | 0.657 |
| Random Ensemble (*k*=2) | 0.873 | 0.827 | 0.080 | 0.600 |
| Random Ensemble (*k*=1) | 0.854 | 0.815 | 0.095 | 0.582 |

across different combinations of $k$ and $r$. As expected, the most significant performance gap between VFL_MoE and Centr_MoE occurs in the case $k = 1$ and $r = 0.25$. This gap narrows as $k$ and $r$ increase, with the slightest difference when VFL_MoE operates with $k = 2$ and $r = 0.75$.

In a direct comparison with the competing model SVFL as shown in Table 3, VFL_MoE demonstrates superior performance when utilizing at least half the training batches that SVFL uses (i.e., $r \geq 0.5$), regardless of the chosen $k$ values. Notably, in terms of the $F1$ metric, VFL_MoE achieves better results than SVFL even with the least amount of training data ($r = 0.25$). Specifically, the $F1$ score improvement for VFL_MoE is 8.3% at $k = 1$ and 8.8% at $k = 2$ when using $r = 0.50$. However, more data ($r = 0.75$) is necessary for VFL_MoE to surpass SVFL in the *FPR* metric. These findings highlight the effectiveness and flexibility of VFL_MoE in scenarios with restricted training data.

Furthermore, when compared to the Baseline model, VFL_MoE consistently outperforms it across nearly all settings of $k$ and $r$ on the *Adult* dataset, similar to the *KronoDroid* dataset findings. The only exception is under the most restrictive conditions ($k = 1$ and $r = 0.25$), where Baseline slightly surpasses VFL_MoE in *ACC* by a marginal 0.6% and significantly in *FPR* by approximately 40%. Despite this, even in such constrained circumstances, VFL_MoE achieves a higher *AUC* and $F1$ score than Baseline, with an increase of 2.3% and 9.8%, respectively.

#### Ablation study

An ablation study was performed to evaluate the individual impacts of various configurations within our methodology. For this study, the variant of VFL_MoE configured to use two experts for each prediction ($k$=2) is regarded as a reference method that effectively balances performance and efficiency. This method was compared to three simplified variants:

1. `Random Ensemble` ($k$=1), where the data-driven gating strategy of the MoE is replaced with a random ensemble mechanism, i.e., for each data instance, one random expert is chosen to classify the tuple. Please note that this simplified variant of the proposed approach actually coincides with the reference `Baseline` method considered so far in our experimental study.
2. `Random Ensemble` ($k$=2), where two experts are chosen at random for each data instance, and the average of the probabilities provided by the two experts is used to classify the instance.
3. `VFL_MoE` ($k$=1), where the gate function of the MoE is used to select just one expert for each data instance.

The outcomes of the ablation study are detailed in Tables 4 and 5, for the KronoDroid and Adult datasets, respectively. No batch reduction factor was applied; all variants processed the full extent of all training batches ($r = 1$) across all experiments.

As expected, for the KronoDroid dataset, integrating the MoE mechanism yields better performance than a pure random strategy used in `Random Ensemble`. Moreover, transitioning from one to two experts further improves results across all variants and performance metrics. Remarkably, FPR registers an improvement of more than 25% when using two experts with the MoE strategy compared to a random selection.

A similar improvement pattern was observed for all metrics for the Adult dataset, though the differences were less pronounced. Specifically, in this case, the FPR improves of about 18% under the same conditions.

## Related work

The intensification of climate changes, primarily attributed to human-driven factors such as fossil fuel consumption, deforestation, and intensive agriculture, is producing serious repercussions on human societies, biodiversity, and ecosystems [1]. This situation has catalyzed the attention to the so-called *Green AI*, an initiative that applies artificial intelligence (AI) techniques to minimize environmental impact and enhance sustainability [54, 55]. Key strategies in Green AI include creating energy-efficient algorithms, designing less power-intensive hardware, and diminishing the carbon footprint of data centres. The rapid advancement of AI, notably the doubling of computational demands for cutting-edge models between 2015 and 2022 [56], underscores the critical need to evaluate AI's environmental implications, challenges, and potential for sustainable development.

As mentioned in Sect. 1, *Federated Learning* (FL), a distinctive computational-distributed paradigm, faces unique challenges within Green AI. Unlike centralized AI systems, which can be more feasibly powered by renewable energy sources (a direction taken by major players like Google, Meta, and Amazon), FL involves end-user devices that draw power from varied local energy sources, each with its own environmental footprint. Consequently, understanding and mitigating the environmental impact of FL, especially in terms of computation and energy expenditure, becomes a crucial endeavour.

The remainder of this section is devoted to provide an overview of previous research work on *Green FL* and on MoE-based approaches to FL, in two separate subsections.

**Green FL**

Recent research started to explore the energy efficiency and carbon footprint of FL solutions. In this context, the term *Green FL* [1, 24] was recently coined to indicate a body of research aimed at reducing the carbon emissions of FL applications while ensuring satisfactory model accuracy performance. Previous work in this domain studied FL's carbon impact either via simulation or in an analytic manner (based on rough simplifying assumptions) [4]. Specifically, the work [3] assessed FL's carbon emissions, but their approach just represents a preliminary study on this topic. The problem of minimizing the energy footprint of client devices in FL networks was investigated in [21–23], yet not in a large-scale perspective. In [1] a data-driven methodology was utilized to evaluate the carbon emissions associated with FL. This was achieved through direct measurements of real-world tasks conducted on a large scale. This enabled extensive examinations of Green FL and an in-depth analysis of emission profiles from all critical elements involved in FL, such as client devices, servers, and the communication infrastructures connecting them.

From a methodological viewpoint, key research topics include optimally balancing energy consumption with training time [14] and developing methods to control total energy usage by adjusting accuracy targets during local training [15]. Moreover, methods for optimizing bandwidth and workload allocation among heterogeneous devices have been proposed [20], along with resource management schemes that leverage device-specific loss functions to enhance accuracy under communication/computational constraints [19].

Complementary to the above-cited research, there are have been proposals focusing on improving communication efficiency and on model compression [57, 58]. For example, the benefit of using model compression and quantization techniques in reducing the carbon emissions of FL training pipelines was explored in [24]. Other approaches to curbing overall carbon emissions exploited gradient quantization (see, e.g., [16] addressing the case of over-the-air FL systems), precision and bandwidth optimization [18], and methods for balancing energy consumption with model loss function optimization [17].

Most of the existing efficiency-aware studies in the field of FL have focused on the Horizontal FL (HFL) setting, where all the parties involved are assumed to own data over an identical feature space. This assumption does not fit many real scenarios, which better suit the less explored paradigm of Vertical FL (VFL), where full information on each data instance can only be obtained by combining all the parties' data features, and the class labels are only known to a single party. Differently than in the HFL case, in the latter setup, the parties need to strictly cooperate across all training iterations, leading to a higher communication cost [25], especially if adopting the common strategy of having one (coordinator/aggregator) party maintain a global version of the model being discovered and of repeatedly updating this mode by using derived data (concerning model parameters or gradients of them) coming from the other parties. To prevent information leakage in this local data exchange, expensive approaches have been proposed that rely on secure protocols involving many peer-to-peer communications and encryption-related data transformation (e.g., based on additive Homomorphic Encryption methods, like in [32]).

If the approach proposed in [25] reduces the number of communications, per training iteration, linear in the number of parties, it entails considerable amounts of exchanged data and encryption-based data processing operations in the end. On the other hand, the simple usage of perturbed local embedding proposed in [59] for data privacy and communication efficiency may strongly undermine prediction accuracy performances. Indeed, in their vertical asynchronous federated learning (VAFL), only the server holds the global model while the local clients train the feature extractors based on their local data. The Federated Block Coordinate Descent (FedBCD) [60] method utilizes a parallel BCD-like approach, enabling each client to perform numerous local updates and exchange information with the other clients to compute their local gradients. However, FedBCD only works with simple machine-learning models like logistic/linear regression.

The generalized problem of conducting the FL process over data with arbitrary splits over both the feature space and the sample space was recently addressed in [61], under the name of Hybrid FL (HBFL). In this paper, an FL algorithm named FedHD is proposed. Since the clients cannot perform local optimization independently under the hybrid data, a tracking variable is introduced to enable them to track the global gradient information and update the model based on their local data. FedHD allows the clients to perform multiple steps of local stochastic gradient descent (SGD), improving communication efficiency.

#### Mixture of experts in FL

A Mixture of Experts (MoE) is an architectural paradigm for distributing a learning process among multiple specialized models. In a Green AI scenario, MoE can offer a different solution for enhancing the efficiency of FL approaches due to their inherent ability to handle data heterogeneity and achieve specialization. This specialization may help improve the overall model's accuracy and reduce communication overheads, as only relevant expert updates must be communicated between nodes and the central server. Furthermore, the modular nature of MoE allows for a flexible adaptation to the diverse nature of FL settings (both HFL and VFL), optimizing resource usage and offering customized learning capabilities.

The usage of MoE in a FL (precisely, HFL) setting has been explored sporadically. For instance, the study in [26] introduces Personalized Federated Learning (PFL) to enhance FL's accuracy while preserving privacy. Initially, a universal public model is trained. Subsequently, each client develops a tailored model utilizing their private data. The final step involves amalgamating the outputs of both private and public models via MoE. This methodology was expanded in [27] by integrating abstract features and modifying the MoE architecture, enhancing decision-making capabilities. However, these methods do not address energy efficiency. Another interesting approach is presented in [28], which introduces *FedMix*, a FL framework designed to address the challenge of data heterogeneity. In FL, data characteristics may differ across users, making a single global model suboptimal. FedMix overcomes this by training an ensemble of specialized models, enabling user-specific selection of the ensemble members. This approach helps mitigate the effects of non-IID data, leading to improved performance over traditional global models. Similarly, [29] also explores handling non-IID data distributions by combining local and global models through an MoE. This method addresses data heterogeneity across

clients by learning a personalized model for each, blending specialized local models with a more generalized global model trained with federated averaging. This approach aims to achieve high performance on specific client data without sacrificing the generalization benefits of a global model, offering an effective balance between specialization and generalization in FL scenarios. Finally, [62] proposes an MoE-based FL strategy where nodes share part of their public datasets with a central coordinator, similar to our approach. However, this study does not focus on efficiency or rely on a VFL architecture.

Applying MoE models to VFL settings remains largely unexplored. To the best of our knowledge, our preliminary work in [30] has been the only existing attempt to bridge this gap. Compared with the framework presented in [30], our current proposal exhibits several points of novely, particularly in terms of privacy preservation, communication efficiency and computational resource savings. Indeed, the approach proposed [30] was exposed to information leakage risks and high communication/computational costs, mainly because it relied on the idea of providing the gate with masked (embedded) versions of all the data shards.

Indeed, if this approach provides some degree of data privacy, transmitting these (potentially) large data embeddings from local nodes to the central node can lead to significant communication overheads. Additionally, using an autoencoder (AE) to generate the embeddings could expose the risk of an “inversion attack" (i.e., an attempt to exploit the AE to reverse the embeddings and recover the original data), which could compromise data privacy. To mitigate this risk, one might think of making each node implement sophisticated data obfuscation strategies (e.g., based on differential privacy [31] or homomorphic encryption [32] methods). However, these methods would increase computational and communication costs considerably, highlighting the critical trade-off between privacy and efficiency in FL systems.

The framework proposed in our current work exploits instead an ad hoc MoE architecture to reach a better trade-off among the diverging goals of privacy preservation, model accuracy and energy saving. Specifically, in this framework, the gate is only provided with a small subset of less-sensitive data (e.g., in a medical context, data encoding generic patient information that does not disclose her specific medical conditions or potential diseases), which is assumed to be already available to all the nodes in the federation.

In addition, in the proposed framework further enhancements are adopted to improve the computational efficiency of the training process. Here, a *Repeated Random Sampling* (RRS) method [33] is exploited, which relies on setting a reduction factor to control the number of data batches used per training epoch. Specifically, a novel random subset of batches is selected at each epoch, enabling a broader exploration of unseen examples compared to a static sampling method. Although more sophisticated data pruning and distillation techniques exist, the chosen method strikes a better balance between effectiveness and computational efficiency.

Finally, our current study includes a comprehensive analysis of how the data reduction factor and the number of selected (controlled by hyper-parameter $k$ parameter) affect accuracy. This analysis offers in-depth insights into the model’s performance under different operational settings, an aspect not explored in [30].

To conclude, the advancements in our current proposal constitute a significant improvement over the approach in [30], offering a more effective balance between the requirements for privacy, efficiency, and classification accuracy. To summarize, to the best of our knowledge and based on the literature survey conducted so far, our proposal has been the first approach to discovering a MoE-based neural classification model in a Vertical Federated Learning setting in a way that ensures a satisfactory balance between privacy and efficiency requirements.

## Discussion and conclusion

In this paper, we introduced a Vertical Federated Learning (VFL) model that utilizes a neural architecture based on the Mixture of Experts (MoE) paradigm. This model is specifically crafted to minimize computation and communication costs within a distributed, privacy-conscious setting involving multiple parties.

We evaluated the proposed framework on real-life datasets, with special attention to a cybersecurity case study concerning a malware detection task. Even when provided with reduced portions (50% and 75%) of the training instances, the proposed method is shown to outperform competitors over different accuracy metrics (in particular, obtaining remarkable FPR reductions of 16.9% and 18.2%, respectively), though the other approaches use all the training data.

*Significance and Implications* The experimental findings presented in this work provide some evidence for the ability of the proposed VFL framework to ensure a satisfactory balance between data/compute efficiency and accuracy performance while trying to ensure privacy preservation by design —in both the training and application steps, the private data of each data-owner node are processed through the expert sub-model deployed in the node, and only the final expert output is shared with the coordinator node. In our opinion, these nice properties make the proposed framework a valuable solution for VFL settings where the goal of discovering an accurate prediction model must be conciliated with energy-saving constraints and strict privacy requirements. In particular, the results of our experimentation showcase the framework's potential to enhance cybersecurity threat detection and prevention in a collaborative yet secure manner.

Another important scenario in which our framework can be profitably used is represented by the healthcare domain, where different parties, such as service providers and institutions, hold sensitive patient information that cannot be shared due to privacy regulations, and different parties often store different kinds of patient data, such as healthcare records, lab results, or imaging scans.

Finally, we hope that the innovative contribution offered, at both technical and experimental levels, by this work will stimulate novel research on the exploitation of MoE-based architectures (or other kinds of modular neural architectures supporting efficient conditional computations) in Federated Learning problems for which the HFL assumption is unsuitable.

*Design choices and limitations* The sparse routing strategy supported by our *VFL_MoE* models allows for cutting computation and communication costs at inference time by selecting and activating only the top-$k$ experts per instance. By contrast, the learning process implemented by the proposed algorithm `VFL_MoE` implements a dense

computation where the output of all the experts is needed for any training instances in order to evaluate the training losses and the required gradients. In principle, also the training of a sparse MoE can be implemented according to a conditional computation scheme where only the output of the experts selected by the gate is calculated for each training instance and used in the back-propagation phase. This clearly entails adopting some effective strategy for approximating the discrete sparse-routing output of the gate in a differentiable way. Common consolidated approaches to this challenging problem consist in: *(a)* resorting to heuristic routing strategies that neglect or roughly approximate the gradient of the gate function, which may well result in slower convergence speed or event in poorly trained models; *(b)* using ST (Straight-Through) estimators [9, 63], which, however, still requires to activate all the experts in the forward pass of each training step, thus resulting in limited efficiency gain; *(c)* exploiting REINFORCE-like schemes, which trade the nice property of being unbiased with prohibitively high variance values (impeding fast convergence) and were recently shown empirically not to work well in MoE learning applications [64]. In the light of the above-mentioned open issues and potential pitfalls of these solutions for sparsely training a MoE, in this work, we have proposed to reduce the computation and communication costs of the learning algorithm by directly shrinking (through a cheap data sampling mechanism) the number of mini-batches (and associated gradient calculation steps) utilized in each training epoch.

Other limitations of the current proposal can be easily overcome by introducing some simple technical modifications. In particular, since conceptually the proposed framework is parametric to the form of the gate and expert sub-models, in order to allow it to deal with other kinds of data modalities (e.g., images, time series, sequences) it suffices to adopt different neural architectures for either implementing these sub-models. As a more efficient alternative, one can think of leveraging off-the-shelf feature extraction methods or embedding models to preliminary convert non-tabular data into a vectorial form, and keep using the same kind of feed-forward gate/expert models as in this work to process the resulting transformed data.

Finally, it is worth noting that the proposed approach can be trivially extended to face multi-class classification scenarios by simply replacing all the one-dimensional output layers with multi-dimensional output layers, while replacing sigmoid transformations with softmax ones. In fact, we pinpoint that our choice of focusing on a binary classification setting mainly served the purpose of making the presentation of the proposal simpler.

*Future work* Moving forward, we plan to investigate introducing advanced privacy-preserving techniques while keeping low the computation and communication costs. Integrating these costs into the model training process could enhance the efficiency of federated learning approaches. In our opinion, adopting a (two-layer) hierarchical MoE architecture is another line of research that deserves being investigated.

We will also explore the opportunity of further improving the efficiency of the learning process by conditionally activating the top-k experts selected by the gate. To this purpose, it looks interesting the solution proposed in [49] to accelerate the training by using ODE-based gradient approximations, as an alternative to current sparse-training methods, exposed to slow-convergence and/or underfitting issues.

**Acknowledgements**
We thank the editors and anonymous reviewers for their helpful comments on earlier drafts of the manuscript. This work was partly supported by (i) research projects FAIR - Future AI Research (PE00000013) and (ii) SERICS (PE00000014), under the NRRP MUR program funded by the European Union - NextGenerationEU. Their support is gratefully acknowledged.

**Author contributions**
All authors equally contributed to this work and reviewed the manuscript.

**Data availability**
The datasets analysed during the current study are available in the github repository: https://github.com/MlkZaq/arabic-short-text-clustering-datasetsand at: https://archive.ics.uci.edu/dataset/2/adult.

## Declarations

**Competing interests**
The authors declare no competing interests.

Received: 31 January 2024 Accepted: 7 May 2024
Published online: 28 May 2024

**References**

1. Yousefpour A, et al. Green federated learning. arXiv preprint arXiv:2303.14604 2023.
2. Huba D, other: Papaya: practical, private, and scalable federated learning. arxiv:2111.04877 2021.
3. Wu C-J, et al. Sustainable AI: environmental implications, challenges and opportunities. CoRR abs/2111.00364 2022.
4. Qiu X. A first look into the carbon footprint of federated learning. Jo Mach Learning Res. 2023;24(129):1–23.
5. Adadi A. A survey on data-efficient algorithms in big data era. J Big Data. 2021;8:24.
6. Albelaihi R, Yu L, Craft WD, Sun X, Wang C, Gazda R. Green federated learning via energy-aware client selection. In: GLOBECOM 2022-2022 IEEE Global Communications Conference, 2022;13–18. IEEE
7. De Rango F, Guerrieri A, Raimondo P, Spezzano G. Hed-fl: A hierarchical, energy efficient, and dynamic approach for edge federated learning. Pervasive Mobile Comput. 2023;92: 101804.
8. Jacobs RA, Jordan MI, Nowlan SJ, Hinton GE. Adaptive mixtures of local experts. Neural Comput. 1991;3(1):79–87.
9. Bengio Y, Léonard N, Courville A. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 2013.
10. Shazeer N, Mirhoseini A, Maziarz K, Davis A, Le Q, Hinton G, Dean J. Outrageously large neural networks: the sparsely-gated mixture-of-experts layer. In: ICLR, 2017;1–17.
11. Lepikhin D, et al. Gshard: Scaling giant models with conditional computation and automatic sharding. arXiv preprint arXiv:2006.16668 2021.
12. Fedus W, Zoph B, Shazeer N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. J Mach Learn Res. 2022;23(1):5232–70.
13. Zhang J, Guo S, Qu Z, Zeng D, Wang H, Liu Q, Zomaya AY. Adaptive vertical federated learning on unbalanced features. IEEE Trans Parallel Distribut Syst. 2022;33(12):4006–18.
14. Tran NH. Federated learning over wireless networks: optimization model design and analysis. In: Proc. of IEEE conference on computer communications (INFOCOM), 2019;1387–1395.
15. Yang Z. Energy efficient federated learning over wireless communication networks. IEEE Trans Wireless Commun. 2020;20(3):1935–49.
16. Zhu G, Du Y, Gündüz D, Huang K. One-bit over-the-air aggregation for communication-efficient federated edge learning: design and convergence analysis. IEEE Trans Wireless Commun. 2020;20(3):2120–35.
17. Feng C. On the design of federated learning in the mobile edge computing systems. IEEE Trans Commun. 2021;69(9):5902–16.
18. Liu P. Training time minimization for federated edge learning with optimized gradient quantization and bandwidth allocation. Front Inf Technol Electron Eng. 2022;23(8):1247–63.
19. Luo B. Cost-effective federated learning design. In: proc. of IEEE conference on computer communications (INFOCOM), 2021;1–10.
20. Zeng Q, Du Y, Huang K, Leung KK. Energy-efficient resource management for federated edge learning with cpu-gpu heterogeneous computing. IEEE Trans Wireless Commun. 2021;20(12):7947–62.
21. Kim YG, Wu C-J. Fedgpo: Heterogeneity-aware global parameter optimization for efficient federated learning. In: Proc of 2022 IEEE Intl. Symp. on workload characterization (IISWC), 2022;117–129.
22. Kim YG, Wu C-J. Autofl: enabling heterogeneity-aware energy efficient federated learning. In: Proc of 54th Annual IEEE/ACM Intl. Symp. on Microarchitecture, 2021;183–198.
23. Abdelmoniem AM, Sahu AN, Canini M, Fahmy SA. Refl: resource-efficient federated learning. In: Proc. of 18th Europ. Conf. on Computer Systems, 2023;215–232.
24. Kim M, Saad W, Mozaffari M, Debbah M. Green, quantized federated learning over wireless networks: an energy-efficient design. IEEE transactions on wireless communications 2023.
25. Xu R. Fedv: Privacy-preserving federated learning over vertically partitioned data. In: Proc. of 14th ACM Workshop on artificial intelligence and security (AISec), New York, NY, USA, 2021;181–192.
26. Peterson DW, Kanani P, Marathe VJ. Private federated learning with domain adaptation. CoRR abs/1912.06733 2019. arXiv:1912.06733

27. Guo B. Pfl-moe: personalized federated learning based on mixture of experts. In: Web and Big Data, pp. 480–486. Springer, Cham 2021.
28. Reisser M, Louizos C, Gavves E, Welling M. Federated mixture of experts. arXiv preprint arXiv:2107.06724 2021.
29. Zec EL, Mogren O, Martinsson J, Sütfeld LR, Gillblad D. Specialized federated learning using a mixture of experts. arXiv preprint arXiv:2010.02056 2020.
30. Folino F, Folino G, Pisani FS, Pontieri L, Sabatino P. A scalable vertical federated learning framework for analytics in the cybersecurity domain. In: Proc. of 32nd Euromicro Intl. Conf. on parallel, distributed, and network-based processing (PDP), p. 2024.
31. Dwork C. Differential privacy: a survey of results. In: Proc. of Intl. Conf. on theory and applications of models of computation, 2008;1–19.
32. Gentry C. Fully homomorphic encryption using ideal lattices. In: Proc. of 41st ACM Sympo. on Theory of Computing, 2009;169–178.
33. Okanovic P, et al. Repeated random sampling for minimizing the time-to-accuracy of learning. arXiv preprint arXiv:2305.18424 2023.
34. Hellmeier M, Pampus J, Qarawlus H, Howar F. Implementing data sovereignty: Requirements & challenges from practice. Proceedings of the 18th international conference on availability, reliability and security 2023.
35. Hummel P, Braun M, Tretter M, Dabrock P. Data sovereignty: a review. Big Data Soc. 2021;8(1):2053951720982012.
36. Esposito C, Castiglione A, Choo K-KR. Encryption-based solution for data sovereignty in federated clouds. IEEE Cloud Comput. 2016;3(1):12–7.
37. Sheikhalishahi M, Saracino A, Martinelli F, Marra AL. Privacy preserving data sharing and analysis for edge-based architectures. Int J Inf Sec. 2022;21(1):79–101.
38. Yang Q, Liu Y, Chen T, Tong Y. Federated machine learning: concept and applications. ACM Trans Intell Syst Technol. 2019;10(2):1–19.
39. Yang L, et al. Vertical federated learning: concepts, advances and challenges. arXiv preprint arXiv:2211.12814v4 2023.
40. Romanini D, et al. PyVertical: a vertical federated learning framework for multi-headed SplitNN 2021.
41. Kaplan J, et al. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 2020.
42. Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Proc. of 27th Intl. Conf. on neural information processing systems - 2014;2:3104–3112.
43. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Commun ACM. 2017;60(6):84–90.
44. Hinton G. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process Mag. 2012;29(6):82–97.
45. Jordan MI, Jacobs RA. Hierarchical mixtures of experts and the em algorithm. Neural Comput. 1994;6(2):181–214.
46. Han S, Mao H, Dally WJ. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 2015.
47. Eigen D, Ranzato M, Sutskever I. Learning factored representations in a deep mixture of experts. arXiv preprint arXiv:1312.4314 2013.
48. Roller S, Sukhbaatar S, Weston J. Hash layers for large sparse models. In: advances in neural information processing systems, 2021;34:17555–17566.
49. Liu L, Gao J, Chen W. Sparse backpropagation for MoE training. In: arXiv:2310.00811 [cs] 2023.
50. Ismail AA, Arik SÖ, Yoon J, Taly A, Feizi S, Pfister T. Interpretable mixture of experts for structured data. arXiv preprint arXiv:2206.02107 2022.
51. Jacobs RA, Jordan MI, Nowlan SJ, Hinton GE. Adaptive mixtures of local experts. Neural Comput. 1991;3(1):79–87.
52. Guerra-Manzanares A, Bahsi H, Nõmm S. Kronodroid: time-based hybrid-featured dataset for effective android malware detection and characterization. Comput Sec. 2021;110: 102399.
53. Platt JC. Fast training of support vector machines using sequential minimal optimization 1998.
54. Schwartz R, Dodge J, Smith NA, Etzioni O. Green AI. Commun ACM. 2020;63(12):54–63.
55. Rolnick D. Tackling climate change with machine learning. ACM Comput Surv (CSUR). 2022;55(2):1–96.
56. Sevilla J, Compute trends across three eras of machine learning. In: Proc. of 2022 Intl. Joint conference on neural networks (IJCNN), 2022;1–8.
57. Vogels T, Karimireddy SP, Jaggi, M. Powersgd: Practical low-rank gradient compression for distributed optimization. Adv Neural Inf Process Syst 2019;32.
58. Rothchild D. Fetchsgd: Communication-efficient federated learning with sketching. In: Proc. of Intl. Conf. on Machine Learning (ICML), 2020;8253–8265.
59. Chen T, Jin X, Sun Y, Yin W. Vafl: a method of vertical asynchronous federated learning. arXiv preprint arXiv:2007.06081 2020.
60. Liu Y. Fedbcd: a communication-efficient collaborative learning framework for distributed features. IEEE Trans Signal Process. 2022;70:4277–90.
61. Su L, Lau VKN. Hierarchical federated learning for hybrid data partitioning across multitype sensors. IEEE Int Things J. 2021;8(13):10922–39.
62. Parsaeefard S, Etesami SE, Leon-Garcia A. Robust federated learning by mixture of experts. CoRR abs/2104.11700 2021. arXiv:2104.11700
63. Liu L, Dong C, Liu X, Yu B, Gao J. Bridging discrete and backpropagation: Straight-through and beyond. In: Proc. of 36th Intl. Conf. on Advances in Neural Information Processing Systems 2024.
64. Kool W, Maddison CJ, Mnih A. Unbiased gradient estimation with balanced assignments for mixtures of experts. 2021.


## Publisher's Note