Ti-skol: A Modular Federated Learning Framework Supporting Security Countermeasure Composition

Divi De Lacour*†, Marc Lacoste*, Mario Südholt†, Jacques Traoré*

*Orange Innovation
†IMT Atlantique, Inria, LS2N
divi1.delacour@orange.com, marc.lacoste@orange.com
mario.sudholt@imt-atlantique.fr, jacques.traore@orange.com

***Abstract*—Federated Learning (FL) is a growing technology that enables training of Deep Learning models on private data. Many FL enhancements have been proposed, notably for better security and privacy. Current architectures and frameworks focus on specific sets of enhancements with little extensibility and do not support composition of enhancements. In this paper, we introduce Ti-skol, an architecture and framework that supports composition of security and privacy countermeasures, including countermeasure incompatibilities. Ti-skol also enables modular management of FL enhancements beyond security, being compatible with most enhancements. Ti-skol is promising to assess the cost of countermeasures, individually or in combination. We evaluate our framework on a use-case of Volunteer Deep Learning – applying Volunteer Computing to reduce the cost of large model training by harnessing idle resources of single machines into the required massive distributed computing power. Experimental results show that Ti-skol is scalable as the network size increases. While adding security countermeasures such as Byzantine protections or secure aggregation substantially increase computing overheads, they do not change their order of magnitude, individually or in combination. This tends to show the practicality of the Ti-skol framework for on-demand FL security.**

***Index Terms*—Federated Learning, Security, Countermeasure Composition, Volunteer Deep Learning**

I. Introduction

In recent years, IoT devices have produced a deluge of data, enabling to train machine learning (ML) models for prediction and classification in a large range of applications [1]. While centralized model training raises privacy concerns, the promise of Federated Learning (FL) [2] is to train joint models without the data leaving the device. A server sends the model to the devices holding the data. Data holders train the model on their local data and send back model updates to the server for aggregation. Only model updates are shared, not raw data. FL systems have been extensively studied [3]. Yet, many security and privacy vulnerabilities have been reported [4]. Threats target mainly privacy, integrity and identity: 1) extracting private information from training datasets [5], [6] or reconstructing raw data [7]; 2) poisoning data samples [8], polluting models through malicious updates [9], [10] or introducing backdoors [11]; and 3) Sybil attacks [12].

An even larger range of countermeasures has been proposed to mitigate such threats with different security guarantees and limitations [4] – with no one-size-fits-all countermeasure to cover the full set of protection requirements. Countermeasures remain difficult to compare and are often not compatible with one another. While a few benchmarking frameworks have been proposed [13], they do not address countermeasure composition – referring to the case when countermeasures are combined on-demand to complement one another.

To enhance FL security, *composition of countermeasures* appears as a focal challenge to: 1) select on-demand the countermeasures meeting the security requirements; 2) handle countermeasure incompatibilities; and 3) assess the cost of countermeasures, individually or in combination. Current frameworks do not meet those goals and do not support modular management of FL security enhancements.

In this paper, we propose TI-SKOL[1], a modular architecture for FL supporting highly expressive customization policies for security countermeasure composition, enabling benchmarking of countermeasures. We propose a framework implementing the architecture and show how security countermeasures can be composed within this architecture, taking as examples Byzantine protection and aggregation integrity. TI-SKOL also enables modular management of composition of FL enhancements beyond security. Indeed, while many FL enhancements have been proposed regarding model quality or performance, current deployment frameworks focus on a specific set of enhancements, but with little extensibility.

To evaluate the TI-SKOL framework, we consider a Volunteer Deep Learning (VDL) use-case. VDL is an application of FL based on the Volunteer Computing paradigm [14] to train DL models. The idea is to use idle computing resources to reduce the cost of large distributed computing tasks.

This paper provides the following contributions: 1) we propose a modular FL architecture that supports on-demand composition of security countermeasures and effective benchmarking in terms of impact on performance; 2) experimental evaluations show that our framework can effectively support

[1]TI-SKOL is the breton word for *school building*. It provides the base infrastructure to organize lectures – centralized, around a teacher or decentralized, rearranging the classroom. Additional security can be added if needed.

countermeasure incompatibilities while preserving modularity, with also good results in terms of performance and scalability. The paper is organized as follows. Section II introduces our reference architecture for FL compatible with legacy enhancements. Section III explains the composition of security countermeasures in our architecture. Section IV presents the TI-SKOL framework. Section V evaluates TI-SKOL scalability and security composition on a VDL use-case. Section VI reviews related work. Section VII concludes.

II. ARCHITECTURE

This section presents the design of our architecture, in terms of key properties, design principles and structure.

A. Properties

We consider the following properties for a FL architecture applicable to real-world systems, regarding *performance* and *customization*.

Performance – The architecture should not hamper application performance nor reliability.

- The architecture should be **scalable**: enhancing performance should be possible by increasing the computing power of nodes (*vertical scaling*); or by supporting a high number of nodes (*horizontal scaling*). FL has shown weaknesses in horizontal scaling [15]. The central server is a bottleneck, limiting client scalability due to available bandwidth and computing resources.
- The architecture should be **resilient** to node failures: FL has shown resilience on the client side, but remains weak on the server side (single point of failure).

Customization – The architecture should be easy to improve by integrating research results and composing them.

- The architecture should be **compatible with legacy** FL systems: it should be expressive enough to support most previous works on FL. It should be compatible with the identified FL design patterns [16].
- **Security countermeasure composition** should be supported to provide *on demand* security and privacy to meet use-case requirements. New countermeasures should be easy to integrate in the architecture.

B. Design Principles

We adopt the following design principles, described next.

Performance – We adopt an architecture which is *aggregator-based* and *fully decentralized*.

- **Scalability** $\Rightarrow$ **Aggregator-based architecture**: the training process should allow multiple concurrent nodes in the aggregation process to reduce the load on the server, as more clients participate [15], [17]. This may be achieved by placing intermediate *aggregators* between the server and clients [15], [17] or with *fully decentralized designs* [18], where participants aggregate the models of their neighbors in the network.
- **Resilience** $\Rightarrow$ **Decentralized architecture**: to tolerate server failures, the architecture should have *no central node* and be fully decentralized [15], [18].

Customization – We adopt an architecture which is *compatible with a reference FL architecture* to enable extensibility and is *modular* to support the composition of countermeasures.

- **Legacy** $\Rightarrow$ **Reference architecture compliance**: the architecture should follow a *reference architecture* such as FLRA [19] to be compatible with most FL enhancements and design patterns [16].
- **Countermeasure composition** $\Rightarrow$ **Modularity**: families of security countermeasures should be identified and managed as components, with specific hooks defined in the architecture. Their incompatibilities should be identified. Seamless replacement of countermeasures with new versions should be supported.

C. TI-SKOL Architecture

We propose TI-SKOL, an architecture based on the previous design principles.

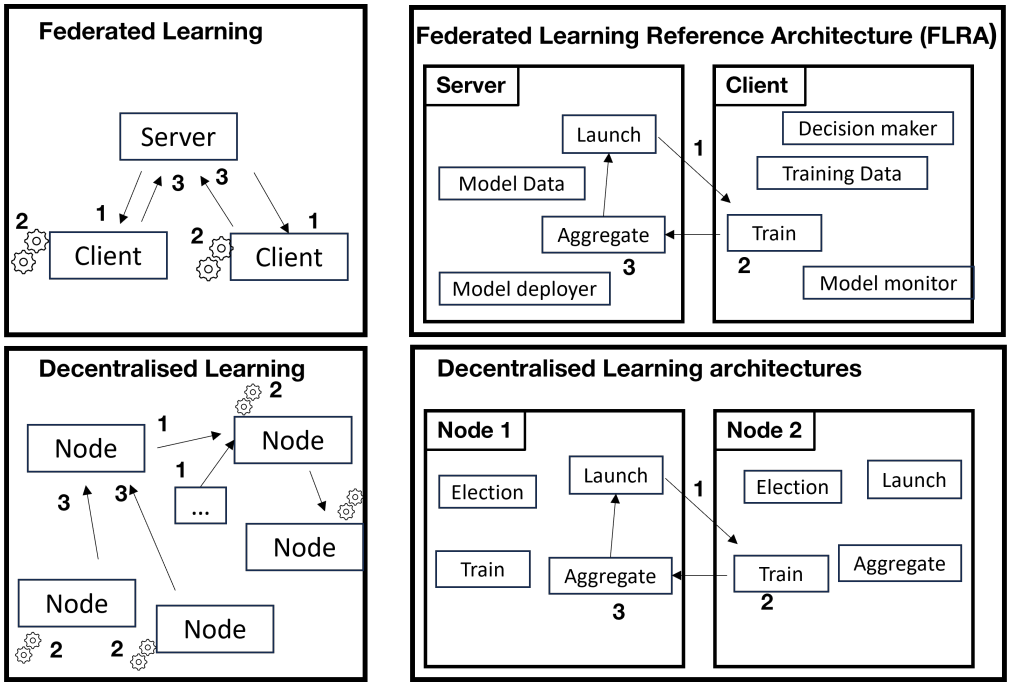
We first review current approaches (see Figure 1). FL and decentralized learning both feature the same training loop (Figure 1a): the model to train is sent to clients (1), trained on local client data (2), and sent back to an aggregator to update the model (3). In FL, the server sends the model and aggregates the response. In decentralized approaches, any participant may perform those tasks.

FLRA [19] is a reference architecture for FL describing the sub-components on client-side and server-side and how they communicate with each other (Figure 1b). Three components are required for training: (1) *Launch* sends the models to train; (2) *Train* trains the models on local data; and (3) *Aggregate* aggregates the received trained models. In decentralized approaches [15], each participant includes all such components, i.e., any component may perform aggregations.

Compatibility with legacy – We design our architecture to be compliant with FLRA [19] and assume the same core components. For decentralized learning, we add the *Election* component to assign to nodes the relevant role in the training process (i.e., client, aggregator, server).

A sample decentralized learning architecture [15] is shown in Figure 2. Similarly to a FL server, the *Aggregation* component features a *Launch* component that starts the training rounds and an *Aggregate* component that receives and aggregates trained models. The *Train* component trains the models similarly to FL clients. Communications between components are performed by a *Network* component, part of each node subcomponent. A *Data* component is in charge of storing *Model* data and *Training Data*.

Countermeasure composition – In our architecture, the ownership relationship between components is structured as a tree. Each branch is dedicated to one of the core system functions. This design ensures that each component is responsible for a specific function without overlap in terms of responsibilities, i.e., only a single component is in charge of aggregation. The



a) b)

Fig. 1: Previous approaches: (a) training methods and (b) reference architectures for federated and decentralized learning

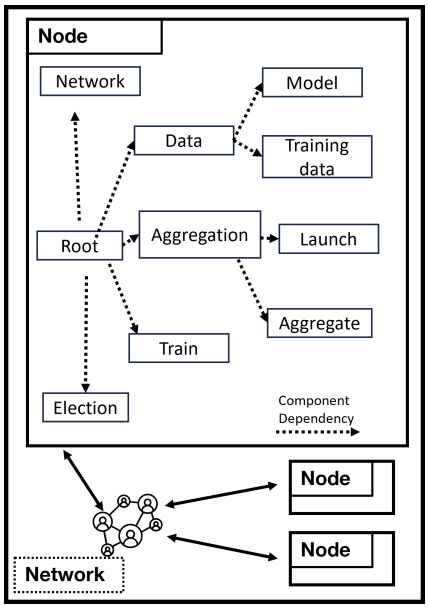


Fig. 2: TI-SKOL architecture

independence of components allows them to be distributed into multiple execution environments for improved performance or isolation.

Scalability and resilience – We extend our architecture to support decentralized learning [15], [18]. Each node can be a client or a server in the training task. Therefore, each node should feature client and server components at any time. An *Election* component is added to reach consensus with peer nodes on their roles [15]. It provides the current role of a node to other components (e.g., allow the *Aggregate* component to launch a training round as server node) and discover other nodes (e.g., find an aggregator in the network for a client).

III. Security Architecture

Our tree-based architecture enables to easily add legacy enhancements by compliance with a FL reference architecture. We now focus on FL security and privacy challenges and applicable countermeasures. We review the challenges, countermeasures and their composition in our architecture. TI-SKOL improves FL security by allowing the implementation and combination of security countermeasures and the isolation of FL software components.

A. Security Challenges

Many threats on DL [20] also apply to FL [4]. We focus on the following key FL security challenges: 1) *integrity*, i.e., training should produce a correct model; and 2) *privacy*, i.e., training data should be kept private.

Threats can be categorized as: *malicious*, where participants may deviate from the protocol; or *honest but curious*, where participants respect the protocol, but try to gain information from legitimate messages.

Privacy. Keeping training data private is a main driver for FL adoption [2], [3]. But assessing the privacy guarantees of FL schemes remains difficult. A common approach for FL

privacy is based on *differential privacy (DP)* [21]. DP provides a probabilistic indistinguishability guarantee for processing a specific data sample during training. This means that a specific training sample cannot be extracted from the trained model – the training data set probability distribution can still be leaked. Privacy requirements may also include confidentiality of the neural network architecture and of weights.

Integrity. FL does not require an exact model. It can even be hard to get the same model every time due to floating-point operations roundings that can vary between different systems. Models can be the target of *backdoor attacks* [11], where model performance is degraded on a specific class of samples. Countermeasures may only filter part of malicious updates sent by clients. Different levels of integrity may be achieved depending of the update impact on the overall model. The challenge is then to evaluate model integrity – model accuracy is a first broad metric.

Identity threats. In *Sybil attacks*, an attacker creates multiple identities to increase its voting power [12]. For FL systems, such attacks are generally not considered, as mitigations such as IP address bans provide a sufficient level of security.

B. Security and Privacy Countermeasures

Privacy countermeasures. Privacy countermeasures include: 1) *differential privacy* to prevent leakage of private data from the model weights; and 2) *secure aggregation protocols* to prevent the aggregating server from accessing the weights of individual models.

DP [21] improves FL privacy by adding random noise to the model weights. DP can be applied by clients to protect their training data from aggregators; or by aggregators to protect training data from other clients during the next training round. DP increases privacy but reduces model accuracy.

Secure aggregation protocols [22] are cryptographic schemes where clients encrypt their model before sending it to the server for aggregation. The server is only able to access the weights of the aggregated model and not of individual models sent by clients. Secure aggregation schemes require high processing power. An algorithm that is scalable, fault tolerant, while supporting Byzantine protection and DP-enabled privacy remains to be found.

Integrity countermeasures. For distributed machine learning, training integrity is closely related to *Byzantine resilience* [23], as malicious models may be introduced in such a decentralized setting. Most solutions against Byzantine clients such as FLAME [24] are based on filtering the most important outliers. They convert the model weights into vectors and compare them, e.g., keeping the median of vectors instead of the average. FLAME also adds noise (DP) to further reduce the effectiveness of backdoor attacks. Other approaches are based on redundancy by making multiple clients perform the same tasks or storing intermediate results. However, they breach training data privacy, as clients have to share their training data for safety checks.

To guarantee *aggregation integrity*, most solutions are secure aggregation schemes [22] that may get in conflict with Byzantine protections. For FL, this challenge is little explored as the central server is considered trusted, under direct control of the organizers.

Privacy and integrity. *Trusted Execution Environments (TEE)* are also part of applicable counter-measures to achieve both privacy and integrity [25], [26]. The TEE encrypted memory and integrity guarantees may help to analyze training data, train or aggregate models in a secure and privacy-preserving manner. TEE remote attestation may also be useful for devices to verify that tasks take place on secure nodes. How to compose TEE with other privacy-enhancing technologies such as distributed protocols, DP, and cryptographic schemes is still an open research challenge.

C. Identity Countermeasures

In [12], two families of protections against Sybil attacks are distinguished: a *central identity management authority* that attributes an identity to each stakeholder, and *Proof of X protocols* that grant voting rights according to the capacity of the smallest participants – e.g., Proof of Work, where voting rights are according to the computing power of a stakeholder.

D. TI-SKOL *Security Architecture*

The tree-shaped organization of components inside a node enables to easily manage *authorizations* and enforce isolation. A unified system for specifying and enforcing component authorizations may be defined from the tree path, i.e., sub-components inherit their authorizations from their parents. This hierarchical approach helps enforcing the principle of least privilege.

Sub-components are more likely to change than their parents, due to more frequent system reconfigurations in the lower-levels. They are therefore considered as less trustworthy. Other sub-components behave as helpers for their parents for core functional sub-tasks. They require similar levels of authorizations as their parents.

This hierarchical approach to component identity management, e.g., for enrolment, helps tracking the component initialization status and increases accountability.

E. Countermeasure Composition in TI-SKOL

Figure 3 shows an extended architecture with the security components (in grey) considered inside each node as part of the training process.

We consider the following security countermeasures: Byzantine protection, secure aggregation and DP. As [22], we consider that a secure aggregation protocol features four components: *init*, *protect*, *aggregate* and *verify*. Byzantine protection replaces the aggregation function, with the models to aggregate as input and the aggregated model as output. DP updates the model weights after aggregation or during training.

The TI-SKOL tree is established by a human expert and captures incompatibilities between countermeasures. For instance,

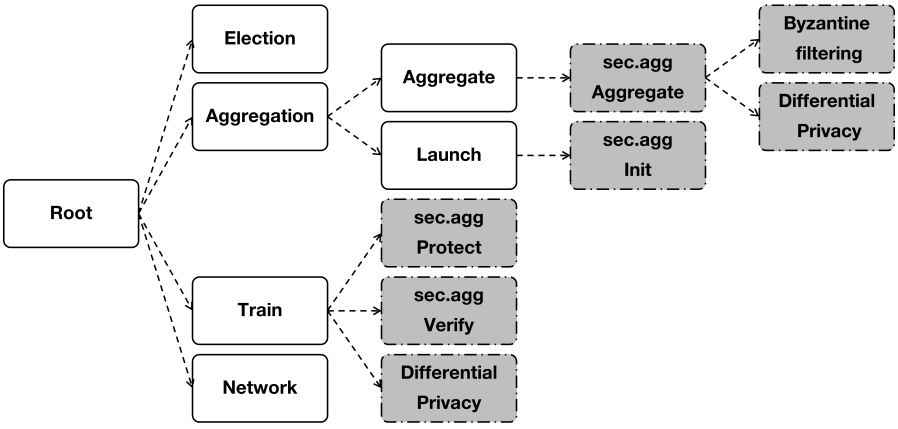


Fig. 3: Countermeasure composition in a node

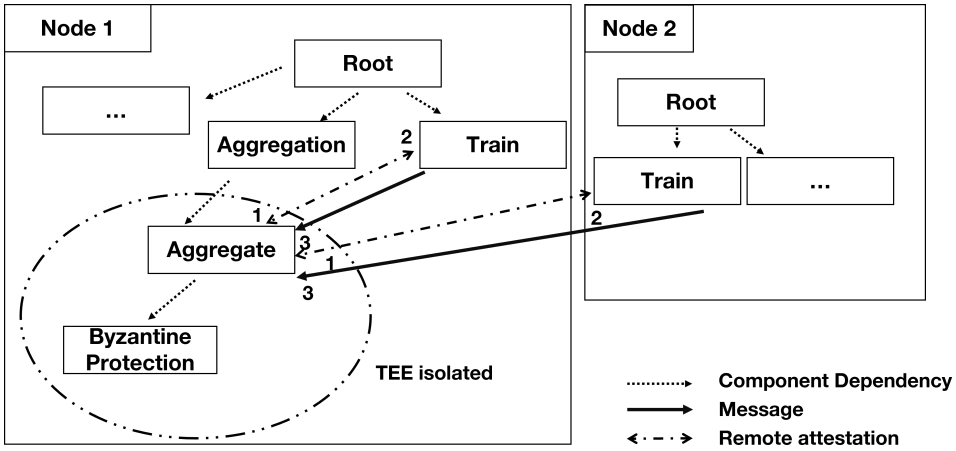


Fig. 4: TEE isolation for the aggregation process

a secure aggregation process may not tolerate Byzantine protections due to privacy guarantees on individual trained models. Conversely, it may allow both DP and Byzantine protection at the same time.

DP can always be added after training or aggregation. Some approaches include DP directly into their aggregation process to improve privacy [22]. In that case, only the aggregated model with DP is visible to the aggregator.

When secure aggregation is compatible with Byzantine protections (e.g., mean aggregation), a training round follows the process shown in Algorithm 1.

A specific security composition (e.g., with/without DP and at which scale, secure aggregation algorithm) aims to meet specific security requirements. Such configuration is independent from the training task and can be re-used for other training tasks or projects. It can either be human or machine generated, taking into account the component compatibilities, performance and security properties for the combination to answer the training requirements.

Algorithm 1 TI-SKOL training round

- **Init**: initialize secure aggregation with **verify** of clients;
- *Aggregation/Launcher*: send `totrain()` to *train* of clients;
- Clients: *train* the model, send it to client **verify**;
- Clients: **protect** trained model, return result to *Aggregation*;
- *Aggregation/Process*: run **Aggregate**;
- **Aggregate**: run **Byzantine protection** if possible;
- **Aggregate**: send result to **verify** of clients.

normal component, **security component**

F. TEE Component Isolation

While TEE are not captured as a specific component of the architecture, they may be simply applied to multiple components. TEEs may help to isolate specific branches in the tree-based TI-SKOL architecture to guarantee integrity and confidentiality. Due to branch independence, this approach enables to limit the trusted computing base (TCB) and enforce

selective component isolation. It also prevents dependence of components on untrusted sub-components. Remote attestation may be used to verify that a component is running in a TEE, performed by other components communicating with it, inside the same node or outside.

Figure 4 illustrates the case where the *Aggregation Process* component is isolated in a TEE to improve aggregation integrity and privacy. Its sub-component in charge of *Byzantine filtering* is also isolated in the TEE. The two training components (in the same node and in another node) both first send a challenge to the aggregation process to verify that it is running in a TEE (1). The aggregation process sends back a proof (2). Once they have verified the proof, the training components notify the aggregation process (3). Remote attestation can be performed only during the first communication. Swarm attestation approaches may help to reduce the attestation overhead.

IV. Implementation

We now describe our implementation. We also develop a use-case of volunteer deep learning for federated learning.

A. TI-SKOL *Implementation*

Figure 5 gives a high-level overview of the TI-SKOL framework. Each participant runs the framework and loads a configuration file and the training data. The application then connects to the cooperation network to participate in the training process – to connect to the server (centralized FL) or to other peers (decentralized learning).

We configure a node using a nested JSON file. A component is described by its configuration and that of its sub-components. This limits the configuration information shared between components to the strict necessary. The initialisation process follows the tree shown in Figure 3, where each component is a js object that creates and configures its sub-components. A component is fully initialized when its sub-components are also initialized. Listing 1 illustrates the configurations of the *Aggregation* component and of its two sub-components: *Launch* and *Aggregate*. The configuration file can include other files for specific parts, e.g., the `Election` component. This design allows reusing well-specified and well-tested configuration for different projects or to create configuration file templates.

Listing 1: Sample aggregator configuration

```
aggregation_launch_config: {
  number_training_steps: 10,
  training_instructions: {
    fit: {
      epochs: 1
    },
    model: "CNN1"
  }
},

aggregation_aggregate_config: {
  secure_aggregation_config: {
    sec_agg: "None",
    byzantine_protection: "FLAME"
  }
}
```

As in most FL systems, training data can be loaded internally from a node or from a public database, e.g., distributed in P2P networks such as IPFS. Nodes that do not have training data may participate in training by providing computing resources.

The framework is implemented in JavaScript for simplicity of deployment – it can be run on any device with a JavaScript run-time (e.g., `Node.js` server, smartphone web browser, video game console). Thus, a single framework may be developed with user-level rights.

JavaScript allows us to run compute-intensive tasks using WASM for improved performance and safety. WASM offers close to native performance. It also improves safety, through type-safe memory isolation for code sandboxing to minimize impact on the operating system or the hypervisor.

Device GPUs may help to accelerate DL model training using frameworks such as TensorflowJS based on CUDA APIs (server side) or WebGL APIs (web browser side).

Communications between nodes are based on `libp2p`[2]. This library for P2P networks supports direct communications between nodes using multiple network protocols (e.g., TCP, WebRTC). It provides a Kademlia Distributed Hash Table to find resources in the network.

Each component includes a *Network* component in charge of communications with other components, inside the same node or between different nodes.

As shown in Listing 2, each component interface includes the component name and a *multiaddr*. These are respectively the path of the component in the tree and the network identifier of the libp2p node. The network identifier includes the communication protocol, IP address, port number, and a hash of the node public key. The interface also includes the methods which may be invoked on a component.

Listing 2: Sample component interface

```
{
  component: '/root/aggregation/[...]',
  multiaddr: '/ip4/127.0.0.1/tcp/52728/[...]',
  handler: 'become_client' // component methods
}
```

This unified communication model facilitates enforcement of component access control policies. For instance, in the *Network* component, a policy could specify that the component is authorized to accept invocations originating only from another specific component of the same node.

V. Experimental Results

We evaluate the TI-SKOL framework scalability and the impact of addition of security countermeasures on performance. We illustrate the framework capabilities with a VDL use-case, including the required security guarantees.

[2]https://libp2p.io/

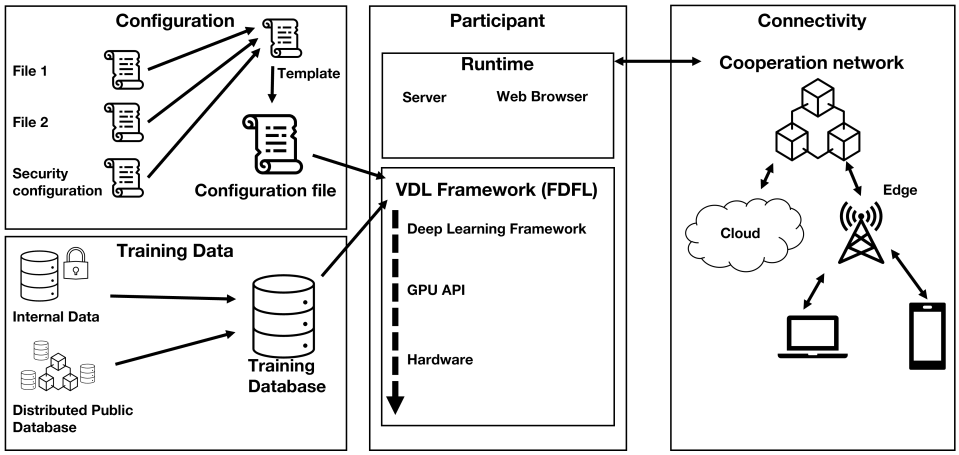


Fig. 5: High-level framework overview

A. Testbed

We run our experiments on a Dell Precision 5750 laptop with an Intel i7, 10th generation, 8 cores, 2.30 GHz CPU and 32 GB RAM, running Windows 10. Framework nodes are run as concurrent `Node.js` processes on the same machine. CPU usage is monitored with the Python library `psutil`. Of course, a single local machine cannot be considered as the optimal setup for a full assessment of scalability, but nonetheless provides insightful results as a first step before moving to a cloud platform.
We train a dense neural network of 101770 parameters in 2 dense layers on the MNIST dataset. This dataset includes 60000 28x28 gray-scale images of handwritten digits to be classified.

B. VDL Use-case

We focus our evaluation on a Volunteer Deep Learning (VDL) use-case. VDL applies the Volunteer Computing (VC) paradigm to train deep learning models to reduce costs. We shortly explain below the main ideas of the use-case.

Volunteer Computing. To reduce costs of large distributed computing tasks, academia introduced *Volunteer Computing* [14]. Explored in projects such as Folding@home [27] or BOINC [28], VC uses idle resources of computers (e.g., CPU scavenging) for scientific tasks (e.g., protein folding simulations, prime number research) that require massive computing power or large amounts of data. Crowdsourcing [29] is a related approach. It uses volunteers to produce data, e.g., to label data sets. The Amazon Mechanical Turk (MTurk)[3] is a commercial example of crowdsourcing against a remuneration.

Volunteer Deep Learning: combining VC and DL. We use the term *Volunteer Deep Learning (VDL)* to refer to configurations when VC is applied to train DL models [30], [31], [32]. VDL leverages federated learning [2] to train DL models on confidential data directly on the devices holding the data.

[3]https://www.mturk.com/

Compared with centralized model training, VDL presents several critical challenges:

- **Synchronization** requires efficient communications and to take into account device heterogeneity.
- **Security** countermeasures are needed to guarantee training integrity and privacy.
- **Deployment** guidelines should be defined to support a high diversity of hardware and software.

While current works have focused on improving performance, security and deployment remain little studied. The VDL use-case requires scalability to support the maximal number of participants. In a first approach, only training integrity is needed. This implies to guarantee aggregation integrity and to protect the system against Byzantine threats.

C. Security Countermeasures

We assessed the composition of countermeasures related to aggregation integrity and Byzantine protection.

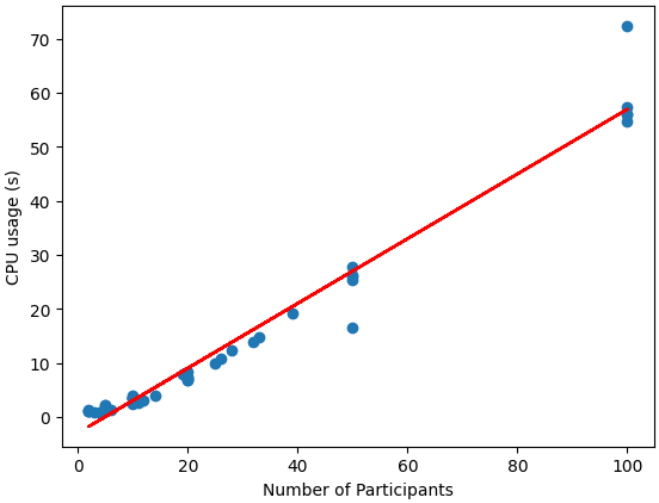


Fig. 6: CPU usage vs. number of training participants

More specifically, we evaluated the following countermeasures for Byzantine protection:

- **None**: Aggregation is performed by averaging the models sent by clients as in traditional FL.
- **Median**: Aggregation outputs the element-wise median of weights of clients.
- **FLAME**: Following the approach of [24], aggregation is based on clustering to eliminate outliers, e.g., using geometric filtering. DP is also used to protect against backdoors.

Regarding the last approach, our implementation did not implement DBSCAN filtering, and as such underestimates CPU usage.

We also evaluated the following countermeasures regarding aggregation integrity:

- **None**: the aggregator averages models without any proof sent to clients.
- **None_Sec**: the aggregator sends an empty proof to clients, automatically accepted.
- **Whole**: each node receives the hashes of the aggregated model and of models from other clients, downloads them and performs again the whole aggregation operation.
- **Hash**: this countermeasure is based on "incremental" *(homomorphic) hashing* [33]. The models are "hashed" by being projected on a smaller space. The verifier receives the hash of the aggregated model and of other client models. It checks the means of the hashes of clients models is identical to the hash of the aggregated model.

The `Hash` aggregation integrity countermeasure is not compatible with `Median` and `FLAME` Byzantine protections as they do not output the means of a set of models.

In this paper, we do not evaluate protection efficiency or compare the security guarantees of countermeasures. We refer the reader to [24] for comparisons for Byzantine protections and to [22], [34] for communication and computation efficiency evaluations for advanced aggregation integrity protections.

`Whole` and `Hash` protections use IPFS to share a model. Thus, every node has already downloaded a model and broadcasts it. This reduces the server load by preventing the server from sending all models to all nodes.

D. Scalability

We evaluate the scalability of TI-SKOL when increasing the number of participants in the training process, spawning multiple processes, each acting as an independent client. We split the dataset so that each client is allocated the same amount of training data. The total amount of data remains the same when increasing the total number of clients. The size of local training datasets only results in being smaller.

Figure 6 shows the total CPU usage of server and clients (in seconds) needed for a training step over the whole dataset. In the case of the MNIST dataset, the training time itself is negligible. We observe a linear increase of the overall CPU overhead of the distribution with the number of participants. Such results tend to show that the framework scales well with the number of participants, as the overhead per participant remains constant [35].

E. Security Countermeasure Composition

We evaluate the CPU overhead when countermeasures are added. For each composition, we run multiple experiments with 10 clients participating in the training process. We compare to a baseline – a configuration with no security countermeasures.

Figure 7a evaluates the total CPU usage for different aggregation integrity protections. We observe a comparable CPU usage with `None_Sec` when an empty proof is sent. This is expected and may reflect a lightweight base cost for remote attestation. `Whole` and `Hash` protections have much higher CPU overheads – up to X4 on average, but remaining practical nonetheless.

Figure 7b shows the total CPU usage for different Byzantine protections. We observe an increased CPU usage for Byzantine protections that remains in the same order of magnitude – up to X2, which tends to show their practicality when used with TI-SKOL.

byz_protection_type sec_agg_type	None	median	FLAME
None	1.00	1.37	0.88
None_Sec	1.24	1.51	0.72
Whole	3.60	4.95	2.97
Hash	3.41	x	x

TABLE I: CPU usage (normalized) vs. countermeasure composition

Table I shows the mean CPU usage when combining Byzantine protections and secure aggregation countermeasures. In the case of homomorphic hashing, only configurations with no Byzantine protection are evaluated, since Byzantine protections are not compatible with the `Hash` integrity protection.

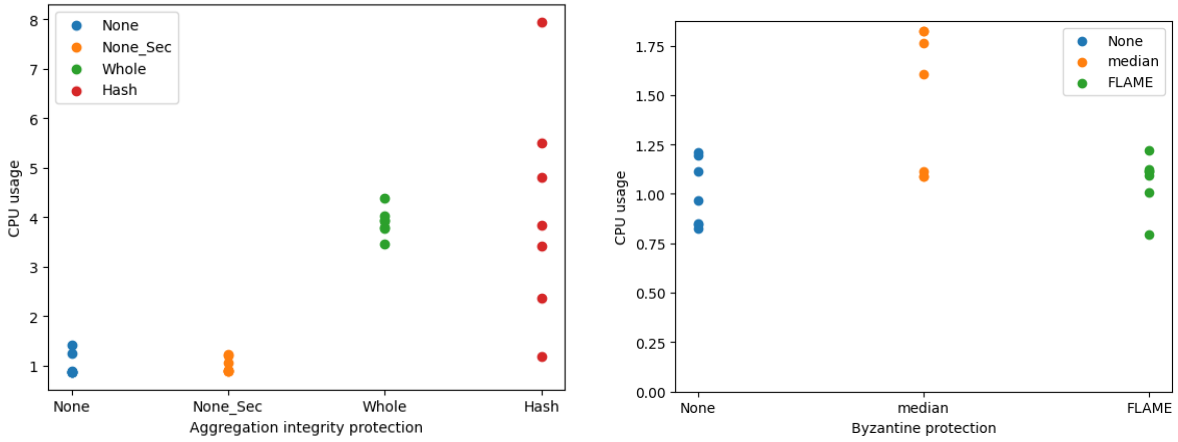
We observe that verifying the integrity of aggregation with that scheme induces far more computing overheads than Byzantine protections – e.g., for `Whole` and `median` countermeasures, the overhead overhead of their combination (3.9) is greater than the result of individual overheads (0.37, 2.6). This confirms that: 1) countermeasures are usually not compositional; and 2) non-negligible overheads may result from composition.

F. Discussion

Scalability and VDL feasibility. We observe a linear increase in the total computing overhead when adding clients. This tends to show that TI-SKOLis scalable when adding FL enhancements.

Such results also encourage the use of decentralized or hierarchical approaches for large-scale model training. We may wonder how this approach scales when increasing the size of the model. Such investigations are deferred for future work.

Security countermeasures composition. Experimental results also show that the security features necessary for practical VDL induce non-negligible CPU overhead, especially secure aggregation and that their combination may exceed the sum



(a) (b)

Fig. 7: CPU usage (normalised): (a) aggregation integrity; (b) Byzantine protections

of individual costs. The security countermeasures composition and management of incompatibility goals for TI-SKOLmay thus be considered achieved in a first step.

Results also show that the different protections stay within the same order of magnitude in terms of computing time (but still with non negligible overhead). Those first experimental results suggest that VDL still remains feasible with an upper bound not exceeding 5 for the overhead – to be confirmed on a testbed VDL platform.

TI-SKOL would show its benefits when there are more computing between synchronizations, e.g., for bigger datasets and with more epochs per training round. TI-SKOL then paves the way for VDL at scale using decentralized learning approaches and neural networks with more parameters.

VI. Related Work

The TI-SKOL approach could be applied to a significant body of work in two broad research areas: FL and VDL.

A. FL Frameworks

Existing architectures and frameworks for FL explore properties such as legacy compatibility, scalability, resilience and security composition. They include FL simulation and production frameworks and decentralized learning frameworks.

FL simulation frameworks. Middleware such as Tensorflow Federated or PyTorch provide highly customizable environments to simulate FL training and add enhancements in a modular manner. Security components can be implemented but their combination often requires a redesign of the whole system. Such middleware are not specifically designed to be deployed for real-life use-cases. As such, scalability and resilience concerns are not usually considered.

FL production frameworks. Middleware such as Flower [36], FedML [37] and ModularFed [38] aim to provide fully deployable frameworks. By design, they handle resilience and scalability for centralized FL and may provide security countermeasures – usually not freely configurable and that need to be placed explicitly in the architecture. Unlike TI-SKOL, such frameworks lack expressivity for enhancing FL due to incompatibility with a reference FL architecture such as FLRA [19].

Decentralized Learning. Architectures and frameworks such as Gossip Learning [39], FDFL [15] and Hivemind [40], [41] provide resilience and scalability. They do not explore the implementation of security countermeasures. Addition of legacy components is more difficult because they diverge from traditional FL. Therefore, such architectures are not compatible with all enhancements. TI-SKOL allows for the extension of FDFL for security composition.

Unlike existing FL frameworks, TI-SKOL provides the customization expressivity of simulation frameworks, including for fully decentralized learning approaches while being deployable. It also remains scalable and supports composition of security countermeasures and management of their incompatibilities.

B. VDL Frameworks

VDL projects include [42], [30], [43], [31], [32], [44]. [31] includes resilience mechanisms and AWS spot instances to reduce computing costs (cloud optimization). Its client-server architecture is implemented with the VC framework BOINC. For training, two VDL projects are up to our knowledge implemented and ready-to-deploy. Hivemind [40], [41] is used to train LLMs [4]. Disco [45] proposes to participate in the training of DL models directly in the web browser, in a federated or decentralized manner – without need to install a software. For inference, Petals [46] runs in a cooperative manner models too large to fit on a single computer.

Unlike current VDL frameworks, TI-SKOL supports composition of security countermeasures for training integrity of VDL tasks.

[4]https://training-transformers-together.github.io/

VII. Conclusion and Next Steps

In this paper, we proposed an architecture and framework supporting the composition of FL security and privacy countermeasures and countermeasure incompatibilities. The framework is scalable and compatible with most FL enhancements. We showed how the framework can be applied to a VDL use-case, highlighting the overhead of countermeasures, individually or in combination, tending to show the framework practicality for on-demand FL security. TI-SKOL enables benchmarking of security countermeasures and their composition for FL frameworks.

Future work includes extending TI-SKOL with a library of standardized security configurations for common FL use-cases. This would be a first step towards an autonomic FL security framework to dynamically adapt defenses to application security requirements.

References

[1] H. Wang et al., "Scientific discovery in the age of artificial intelligence," *Nature*, vol. 620, no. 7972, pp. 47–60, 2023.

[2] B. McMahan et al., "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.

[3] P. Kairouz et al., "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[4] E. Hallaji et al., "Federated and Transfer Learning: A Survey on Adversaries and Defense Mechanisms," in *Federated and Transfer Learning*. Springer, 2023, pp. 29–55.

[5] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 3–18.

[6] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 691–706.

[7] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *NeurIPS*, 2019.

[8] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*. Springer, 2020, pp. 480–501.

[9] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to Byzantine-Robust federated learning," in *29th USENIX Security Symposium*, 2020, pp. 1605–1622.

[10] Y. Fraboni, R. Vidal, and M. Lorenzi, "Free-rider attacks on model aggregation in federated learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PLMR, 2021, pp. 1846–1854.

[11] H. Wang et al., "Attack of the Tails: Yes, You Really Can Backdoor Federated Learning," in *NeurIPS*, 2020.

[12] J. R. Douceur, "The Sybil Attack," in *Peer-to-Peer Systems*. Springer Berlin Heidelberg, 2002, pp. 251–260.

[13] S. Han et al., "FedMLSecurity: A Benchmark for Attacks and Defenses in Federated Learning and Federated LLMs," in *30th SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024.

[14] T. M. Mengistu and D. Che, "Survey and Taxonomy of Volunteer Computing," *ACM Comp. Surv.*, vol. 52, no. 3, pp. 59:1–59:35, 2019.

[15] D. De Lacour, M. Lacoste, M. Südholt, and J. Traoré, "Towards Scalable Resilient Federated Learning: A Fully Decentralised Approach," in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops*, 2023, pp. 621–627.

[16] S. K. Lo, Q. Lu, L. Zhu, H.-Y. Paik, X. Xu, and C. Wang, "Architectural Patterns for the Design of Federated Learning Systems," *Journal of Systems and Software*, vol. 191, p. 111357, 2022.

[17] K. Bonawitz et al., "Towards Federated Learning at Scale: System Design," in *Second Conference on Machine Learning and Systems (SysML)*, 2019.

[18] I. Hegedűs, G. Danner, and M. Jelasity, "Gossip Learning as a Decentralized Alternative to Federated Learning," in *Distributed Applications and Interoperable Systems (DAIS)*, 2019, pp. 74–90.

[19] S. K. Lo et al., "FLRA: A reference architecture for federated learning systems," in *European Conference on Software Architecture*, 2021.

[20] M. Xue et al., "Machine Learning Security: Threats, Countermeasures, and Evaluations," *IEEE Access*, vol. 8, pp. 74 720–74 742, 2020.

[21] M. Abadi et al., "Deep Learning with Differential Privacy," in *2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 308–318.

[22] M. Mohamad, M. Önen, W. Ben Jaballah, and M. Conti, "SoK: Secure aggregation based on cryptographic schemes for federated learning," in *23rd Privacy Enhancing Technologies Symposium (PETS)*, 2023.

[23] R. Guerraoui, N. Gupta, and R. Pinot, "Byzantine Machine Learning: A Primer," *ACM Computing Surveys*, vol. 56, no. 7, pp. 1–39, 2023.

[24] T. D. Nguyen et al., "FLAME: Taming Backdoors in Federated Learning," in *31st USENIX Security Symposium*, 2022, pp. 1415–1432.

[25] S. Prakash, H. Hashemi, Y. Wang, M. Annavaram, and S. Avestimehr, "Byzantine-Resilient Federated Learning with Heterogeneous Data Distribution," *arXiv:2010.07541*, 2021.

[26] I. Messadi et al., "SplitBFT: Improving Byzantine Fault Tolerance Safety Using Trusted Compartments," in *Middleware*, 2022.

[27] A. L. Beberg et al., "Folding@home: Lessons from eight years of volunteer distributed computing," in *IPDPS*, 2009.

[28] D. Anderson, "BOINC: A Platform for Volunteer Computing," *J. Grid Computing*, vol. 18, p. 99–122, 2020.

[29] M. Hirth et al., "Anatomy of a crowdsourcing platform - using the example of microworkers.com," in *IMIS*, 2011.

[30] M. Ryabinin and A. Gusev, "Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts," in *NeurIPS*, 2020.

[31] M. Atre, B. Jha, and A. Rao, "Distributed Deep Learning Using Volunteer Computing-Like Paradigm," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021.

[32] M. D. et al., "Distributed deep learning in open collaborations," in *NeurIPS*, 2021.

[33] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental Cryptography: The Case of Hashing and Signing," in *Annual International Cryptology Conference (CRYPTO)*, 1994.

[34] X. Guo et al., "VeriFL: Communication-Efficient and Fast Verifiable Aggregation for Federated Learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1736–1751, 2021.

[35] E. A. Luke, "Defining and measuring scalability," in *Scalable Parallel Libraries Conference (SPLC)*. IEEE, 1993.

[36] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, H. L. Kwing, T. Parcollet, P. P. d. Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

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

[38] M. Arafeh, H. Otrok, H. Ould-Slimane, A. Mourad, C. Talhi, and E. Damiani, "Modularfed: Leveraging modularity in federated learning frameworks," *internet of Things*, vol. 22, p. 100694, 2023.

[39] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[40] Learning@home team, "Hivemind: a Library for Decentralized Deep Learning," 2020, accessed: 2023-09-22. [Online]. Available: https://github.com/learning-at-home/hivemind

[41] A. Borzunov et al., "Training transformers together," in *NeurIPS 2021 Competition and Demonstration Track*. PMLR, 2022.

[42] M. Ryabinin et al., "Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices," in *NeurIPS*, 2021.

[43] E. Kijsipongse, A. Piyatumrong, and S. U-ruekolan, "A hybrid GPU cluster and volunteer computing platform for scalable deep learning," *The Journal of Supercomputing*, vol. 74, no. 7, pp. 3236–3263, 2018.

[44] Z. Tang et al., "FusionAI: Decentralized Training and Deploying LLMs with Massive Consumer-Level GPUs," in *LLM-IJCAI*, 2023.

[45] E. M. Learning and O. Laboratory, "DISCO - Distributed Collaborative Machine Learning," 2023, accessed: 2023-09-22. [Online]. Available: https://github.com/epfml/disco

[46] A. Borzunov et al., "Petals: Collaborative Inference and Fine-tuning of Large Models," in *61st Annual Meeting of the Association for Computational Linguistics*, 2022.