Sequoia: An Accessible and Extensible Framework for Privacy-Preserving Machine Learning over Distributed Data

KAIQIANG XU, Hong Kong University of Science and Technology, Hong Kong
DI CHAI, Hong Kong University of Science and Technology, Hong Kong
JUNXUE ZHANG, Hong Kong University of Science and Technology, Hong Kong
FAN LAI, University of Illinois Urbana-Champaign, United States
KAI CHEN, Hong Kong University of Science and Technology, Hong Kong

Privacy-preserving machine learning (PPML) algorithms use secure computation protocols to allow multiple data parties to collaboratively train machine learning (ML) models while maintaining their data confidentiality. However, current PPML frameworks couple secure protocols with ML models in PPML algorithm implementations, making it challenging for non-experts to develop and optimize PPML applications, limiting their accessibility and performance.

We propose Sequoia, a novel PPML framework that decouples ML models and secure protocols to optimize the development and execution of PPML applications across data parties. Sequoia offers JAX-compatible APIs for users to program their ML models. It uses a compiler-executor architecture to automatically apply PPML algorithms and system optimizations for model execution over distributed data. The compiler in Sequoia incorporates cross-party PPML processes into user-defined ML models by transparently adding computation, encryption, and communication steps with extensible policies. The executor efficiently schedules code execution across multiple data parties, considering data dependencies and device heterogeneity.

Compared to existing PPML frameworks, Sequoia requires 64%-92% fewer lines of code for users to implement the same PPML algorithms, and achieves 88% speedup of training throughput in horizontal PPML.

CCS Concepts: • **Software and its engineering → Compilers**; **Abstract data types**; **Development frameworks and environments**; • **Computing methodologies → Distributed computing methodologies**.

Additional Key Words and Phrases: Privacy-preserving machine learning (PPML); secure computation; distributed data; compiler-executor architecture; training throughput

ACM Reference Format:
Kaiqiang Xu, Di Chai, Junxue Zhang, Fan Lai, and Kai Chen. 2025. Sequoia: An Accessible and Extensible Framework for Privacy-Preserving Machine Learning over Distributed Data. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 74 (February 2025), 27 pages. https://doi.org/10.1145/3709742

1 Introduction

Recent years have seen a rise in public concern over personal privacy, and governments globally regulate personal data sharing and usage (e.g., GDPR [30]), driving the demand for secure computing solutions in regulated sectors like banking, health, and insurance. Privacy-preserving machine

Authors' Contact Information: Kaiqiang Xu, Hong Kong University of Science and Technology, Hong Kong, Hong Kong, kxuar@cse.ust.hk; Di Chai, Hong Kong University of Science and Technology, Hong Kong, Hong Kong, dchai@cse.ust.hk; Junxue Zhang, Hong Kong University of Science and Technology, Hong Kong, Hong Kong, zjx@ust.hk; Fan Lai, University of Illinois Urbana-Champaign, Urbana-Champaign, IL, United States, fanlai@illinois.edu; Kai Chen, Hong Kong University of Science and Technology, Hong Kong, Hong Kong, kaichen@cse.ust.hk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 2836-6573/2025/2-ART74
https://doi.org/10.1145/3709742

learning (PPML) has become the key technological enabler for organizations to train ML models collaboratively using distributed datasets without risking data leaks. For example, hospitals can use PPML to jointly train medical image diagnosis models without sharing patient data [40].

PPML experts have designed many efficient secure computation protocols based on techniques like homomorphic encryption [13, 19] and secret sharing [26, 33]. However, the real-world situation is that, the adoption of PPML in today's ML applications is still in a very early stage, because integrating PPML algorithms into these ML applications presents a knowledge barrier. For example, NVFlare [7] users need to call 15 internal APIs [3] to perform secure aggregation in PPML, while adding a new PPML algorithm requires further API change and application porting. We found that developing PPML applications correctly and efficiently over data across multiple parties requires understanding *when* and *how* to utilize security-related features within existing PPML frameworks.

As a result, although there have been significant advancements in security and ML algorithms, the PPML frameworks – responsible for taking ML model definition and executing PPML computation across diverse data, hardware, and network conditions – fall short in addressing both *upper-level user accessibility* and *lower-level extensibility* for new and existing PPML algorithms and optimizations, becoming a narrow waist of the software stack. Our view is that PPML frameworks should remove the complexity of PPML integration from the ML application level.

Existing PPML frameworks have limitations in their architecture, which causes secure protocols and ML models to be tightly coupled in PPML applications. This hinders both the development of new applications by ML users and the addition/optimization of algorithms by PPML experts. We briefly introduce existing PPML frameworks under three categories and leave more in-depth discussions in §2.3.

- **Client-Server Model.** TensorFlow Federated [11], OpenFL [25], and Flower [14] utilize a client-server architecture where clients train local models, and the server performs parameter aggregations. However, PPML algorithms [26, 33] that require inter-party peer-to-peer communications are not suitable for this client-server computation model.
- **API Frameworks.** Frameworks such as NVFlare [7], PySyft [9], FATE [1] and MP-SPDZ [42] each develop unique APIs and workflow abstractions, which can be challenging for users to adopt in their applications. This complexity arises from the need to learn and correctly utilize the programming interfaces of both ML models and secure computation protocols.
- **One-off Implementations.** PPML researchers also build one-off frameworks and libraries tailored to specific PPML algorithms, such as VF^{2}Boost [28] and REDsec [26], offering point solutions. However, these frameworks lack extensibility, leading to missed opportunities for new secure protocols in PPML applications. They also push standard system and security challenges, such as batching [38, 75] and pipelining [55], to individual implementations.

Additionally, optimizations in existing work are often handcrafted without low-level abstractions like tasks and operators, therefore, optimization opportunities that already exist in ML systems (e.g., task graph scheduling [32], operator acceleration [76]) may not be fully exploited. This causes software fragmentation and reduces the flexibility and efficiency of PPML applications.

The need for the division of labor between ML and PPML experts drives new requirements for novel PPML frameworks:

(1) *The front-end API* needs to be expressive and flexible for non-experts, and should abstract away many of the complexities in underlying PPML algorithms, including hiding the model-independent secure protocols by transparently fusing cryptographic operations such as Homomorphic Encryption (HE) [2, 12] into ML models.

(2) *The back-end executor* must be optimized for efficiency, with considerations of the interleaved data dependencies and heterogeneous computing devices across multiple data parties, and efficiently schedule the execution of computation and communication to minimize straggler blocking and achieve higher global speed.

In this paper, we propose Sequoia, a general-purpose PPML framework that decouples secure computation protocols and ML models, allowing flexible combination and independent optimization of the secure protocol and ML model inside any PPML application. Through a compiler-executor architecture, Sequoia's compiler transparently transforms the user-defined ML model into an optimized, PPML-enabled program. The executor efficiently schedules code execution across multiple data parties, managing data dependencies and device heterogeneity.

Sequoia has key innovation in three aspects:

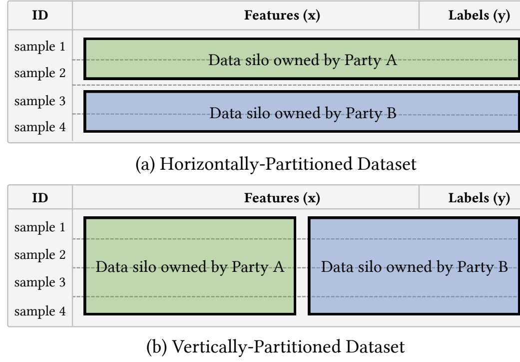
PPML Intermediate Representation (PPIR) (§3). PPIR is a novel approach to building extendable compiler infrastructure and execution environment for PPML. PPIR effectively captures the complex semantics of PPML algorithms with distributed data types and extendable PPML-specific operations, such as secure aggregation and cryptographic operations. The upstream compiler generates PPIR with secure computations incorporated into the user-defined ML model, while the downstream executor focuses on optimizing PPIR execution in the target environment. PPIR decouples model expression from PPML execution, and also facilitates the connection to existing lower-level compilers (e.g., MLIR [48], XLA [63]) to generate optimized executables.

Front-end API and Compiler (§4). Sequoia's high-level API abstracts away almost all PPML-related aspects, requiring only model and data distribution specification from the user. This makes programming PPML-enabled ML models with Sequoia similar to working with centralized models in frameworks like PyTorch. To achieve this, Sequoia's compiler analyzes the user-defined ML model's syntax tree, identifies operations needing cross-party computation by abstractly evaluating data shapes, and automatically applies secure computation protocols in PPML algorithms. Furthermore, Sequoia is *not* tied to specific PPML algorithms; it offers an extensible compiling mechanism where PPML algorithms are added as transformation policies.

Back-end Executor (§5). Sequoia's executor constructs a computation graph from PPIR and finds an efficient execution schedule to optimize global speed. Sequoia has the following key features to handle PPML computations, which have interleaved data dependency patterns spanning both cross-party and intra-party contexts. (1) Sequoia efficiently schedules and executes the computation and communication tasks in each data party with a decentralized scheduler. (2) Sequoia supports two levels of compute nodes, distributing PPML computations efficiently across and within parties using different parallelization strategies.

To summarize, Sequoia makes the following contributions with a novel system design using a compiler-executor architecture that decouples secure computation from ML model computation:

- For ML users, Sequoia enables the expression of multi-party models without enforcing security expertise, facilitating the division of labor in PPML and enhancing PPML accessibility.
- For PPML developers, Sequoia addresses software fragmentation with an extensible compiling mechanism that allows adding new PPML algorithms and connecting to lower-level compilers, without changing user programs.
- Sequoia proposes a decentralized cross-party scheduling algorithm under the two-level compute node hierarchy (intra- and inter-party), improving PPML efficiency in distributed model execution.
- Evaluation shows that, when compared to popular baselines, Sequoia requires 64%-92% fewer lines of code and achieves up to 88% training throughput speedup in horizontal PPML.



(a) Horizontally-Partitioned Dataset

(b) Vertically-Partitioned Dataset

Fig. 1. The two paradigms of cross-silo PPML are based on how data is distributed across data parties. These data distributions commonly occur in real-world scenarios, such as hospitals collaborating to predict patient health conditions (see examples in §2.1).

2 Background and Related Work

Privacy-preserving machine learning (PPML) is a technique for training Machine Learning (ML) models on decentralized data while maintaining data confidentiality.

2.1 Cross-silo PPML

This paper focuses on optimizing PPML under the cross-silo setting [37, 41, 75]. A data silo is a collection of data controlled by a single organization (e.g., institution, company, etc.), which remains separate from other organizations due to laws and regulations [30].

Cross-silo PPML can be particularly useful for organizations that collect sensitive data from their users, such as healthcare providers or financial institutions. These organizations can collaboratively train accurate and robust models while ensuring that their users' data remain secure and private within their premises.

Problem Formulation and Data Distribution. In cross-silo PPML, multiple data silos collaborate to train a global model without directly sharing their data. The learning objective aligns with the standard machine learning optimization process, formulated to find the optimal parameters θ^* that minimize a given loss function:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} L(\theta; x_i, y_i) \tag{1}$$

Here, θ denotes the parameters of the model being trained, and $L(\theta; x_i, y_i)$ represents the loss function for a sample (x, y), measuring the discrepancy between the predicted and the true label.

$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ denote the dataset with N samples. In Equation 1, x_i represents the feature vector, and y_i is the target label for the i-th sample. In cross-silo PPML, each data silo (referred to as "data parties") owns a partition of the dataset, collaborating with other parties to train a global model over the distributed data.

This collaborative learning process depends on specific PPML algorithms [2, 34, 74] and typically involves secure computations to protect the privacy of each silo's data. Based on how data distribution across participating data parties (Figure 1), PPML schemes can be categorized into horizontal PPML and vertical PPML [73].

Algorithms for Horizontal PPML. In horizontal PPML (Figure 1a), data parties have different sample spaces but share the same feature space. Each party controls the labels of its samples. For

example, hospitals with electronic health records data may have similar data structures but different patient information, they can collaborate to train a global model to predict the change of a patient's health condition based on their health records.

Data parties train local models using their own samples and perform cross-party model parameter aggregation [44, 50, 57, 65] to update the model globally. To protect data privacy, some methods [15, 68] employ cryptographic techniques, such as Secret Sharing and Homomorphic Encryption (HE) [12], to aggregate over the received ciphertexts for the global model update without disclosing their local results.

Algorithms for Vertical PPML. In vertical PPML (shown in Figure 1b), data parties in vertical PPML settings have different feature spaces over the samples. For example, an insurance company can collaborate with a hospital to predict the patient's insurance claims based on the patient's health records, while the health records remain private and are not shared between the two parties, ensuring compliance with privacy regulations and preventing direct access to sensitive data.

Simple parameter aggregation methods used in horizontal PPML are inadequate for vertical PPML, because each participant only sees part of the model parameters corresponding to its local feature dimensions during training. Vertical Linear Regression [43, 74] uses HE to encrypt intermediate results and exchange them with other data parties. Split Learning [34] divides a neural network into two parts: the front end, which is trained on the node where the feature data is located, and the back end, which is trained on where the label (target) data is located, therefore, a global model is trained across data parties without transmitting any local label features.

PPML Inference. In horizontal PPML, each party obtains a local model after training and can independently run inference on its local data as with traditional machine learning models, without needing secure computations. In contrast, inference in vertical PPML typically requires cross-party secure computations due to two factors: (1) The final model may be split across parties during training, requiring collaborative inference to make predictions on new data. (2) Even if the final model is made centralized after training, inference may still involve multiple parties since the input features for new data may be distributed among them. This paper focuses on the training process in PPML, as inference can be considered as a part of the computations in training, especially for vertical PPML.

2.2 Decentralized and Federated Learning

While PPML is inherently distributed due to multiple data parties collaboratively training a model, *decentralized learning* (and its subset, federated learning) follows different computational paradigms.

In decentralized learning, each worker m holds a local dataset $\mathcal{D}_m$ and performs local training to minimize loss function with local model parameter θ_m^*:

$$\theta_m^* = \underset{\theta_m}{\operatorname{argmin}} \frac{1}{|\mathcal{D}_m|} \sum_{i=1}^{|\mathcal{D}_m|} L(\theta_m; x_i, y_i) \tag{2}$$

After local optimizations, the workers share their model updates. Prior work [29, 71] explores communication optimizations to reduce overhead and improve efficiency in model synchronization.

Federated Learning (FL), a specific form of decentralized learning, is designed to train machine learning models across numerous decentralized devices, often referred to as "clients". Initially proposed by Google, FL was developed as a cloud-based scheme to train a predictive model for keyboard input on Android devices, with data distributed across numerous edge devices (e.g., smartphones) connected via the internet. We make a special note that some recent literature refers to *cross-silo FL* [37, 75], which actually aligns with the definition of cross-silo PPML in §2.1. Here, we use FL in its original sense as proposed by Google, referring to its cross-device environments.

Unlike traditional decentralized learning, FL introduces additional challenges due to the heterogeneous nature of its clients, which vary in computational resources, connectivity, and dataset sizes. Furthermore, while FL shares privacy considerations with PPML, it requires distinct system optimizations, such as participant selection [47], and security measures, including data integrity validation [51, 60], to address the decentralized and dynamic environment in which it operates.

2.3 Related Work: Existing Systems for PPML

PPML frameworks facilitate the PPML process in which the data owners collaboratively train a model without exposing their data to others. We categorize existing frameworks as follows.

Client-Server Architectures. Many of the existing frameworks employ the classic client-server architecture, where the clients train the local model with local data, and the server is in charge of aggregations. For example, TensorFlow Federated [11], OpenFL [25] and Flower [14] implement server-client APIs for local clients to send model information (e.g., parameter or gradients) to the server.

The client-server model lacks architectural support and optimization for inter-party dependencies that require peer-to-peer communication, making it unsuitable for certain secure computations such as ABY [23]. Moreover, since the intermediate results sent to the central server can reveal information about the parties' local data, the client-server model is also not suitable in scenarios where end-to-end security is required [27].

API Frameworks. Other existing frameworks, including PySyft [62], NVFlare [7], Tensorflow Encypted [10], and FATE [1], provide a fixed set of functionalities through APIs for user program integration. They introduced their own workflow abstractions and APIs for cross-silo PPML: users are required to implement both PPML algorithms and ML model definitions using their specific internal APIs and workflows.

For example, NVFlare requires the use of 15 internal API classes [8] in the ML model definition for horizontal PPML, and adding a new PPML algorithm requires further API change and user program adoption/porting. This approach leads to a steep learning curve and causes fragmentation from a software architectural perspective.

One-off Implementations. In response to the limited extensibility found in the aforementioned frameworks, PPML researchers and practitioners today build one-off frameworks or libraries tailored to specific PPML algorithms. For example, VF^{2}Boost [28] and SecureBoost [18] were created specifically for gradient boosting decision tree (GBDT) algorithms [49], REDsec [26] for FHE-based neural network inference, PEA [61] for differential privacy-based multi-party learning.

While it is technically possible to implement new PPML algorithms by integrating existing ML and cryptographic libraries, this approach imposes an engineering barrier on PPML application development, as the need to address standard system and security challenges such as scheduling, data movement, and cryptographic operations falls on each individual implementation.

2.4 Motivations of Sequoia

PPML applications require a complex integration of secure computations and ML models. We aim to achieve two primary objectives:

- Creating a user-friendly framework that simplifies the integration of secure computations and ML models, and improves execution efficiency through system-level optimizations.
- Designing an extensible mechanism that can support a wide range of PPML algorithms, both current and future, ensuring accessibility and efficiency in deploying these PPML algorithms.

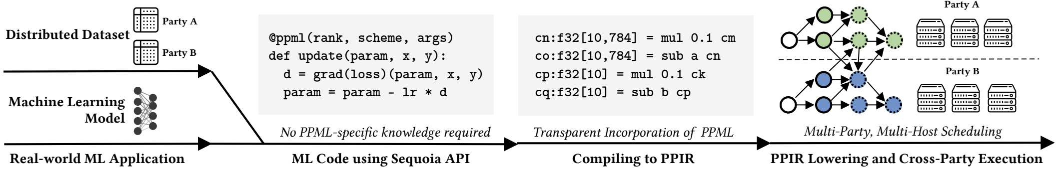


Fig. 2. Overview of Sequoia's workflow. Sequoia's API abstracted away the complexities in PPML algorithms. Sequoia employs a compiler-executor architecture: the compiler (§4) transforms the user's ML code (including model definition, forward pass and training code, see the example in §4.1) into an optimized, PPML-enabled intermediate representation (PPIR, §3), and the executor (§5) lowers PPIR to machine code and executes inside each data party.

3 PPML Intermediate Representation

Intermediate Representation for PPML (PPIR) is a novel approach to constructing a reusable and extendable compiler infrastructure and execution environment for PPML. PPIR serves as a backbone for Sequoia, which expresses cross-party model computations and integrates Sequoia into existing compiler infrastructures.

Figure 2 depicts an overview of the compiler-executor workflow in Sequoia and shows where PPIR fits in. The compiler translates the user-defined ML model into an optimized PPIR, the executor generates code for each data party and schedules for efficient execution on the computation graph.

To represent complex PPML algorithms, PPIR provides enhanced expressiveness and flexibility in its definition:

Unified Data Type for Distributed Data. One of the key strengths of PPIR is its ability to represent the distributed data layout that is unique in PPML settings. PPIR can view data that is distributed across multiple parties as a single variable and apply high-level operations on these distributed variables. This allows for a more efficient and expressive representation of PPML algorithms, and the client operations can be derived from the high-level representation at a later stage.

Target-Specific Operations. Another key feature of PPIR is its ability to utilize target-specific optimizations, including accelerator-specific operations or basic cryptographic operations. To achieve this, a two-stage PPIR compilation is used to effectively capture the complex semantics of PPML algorithms, enabling efficient low-level optimization on target platforms.

3.1 PPIR Syntax and Distributed Data Types

PPIR Syntax. PPIR's syntax is explicitly typed, functional, and follows A-normal form (ANF) principles [64]. Its structure draws inspiration from the successful *jaxpr* format [16]. The formal grammar of PPIR, shown in Figure 3, comprises three segments: input (*lambda*), operations (*let*), and output (*in*).

The layout of data in PPML is distributed across multiple data parties to train the same ML model. The distribution of data has two levels, cross- and intra-party. The cross-party distribution is unique to PPML algorithms, where data from one party cannot be revealed to any other parties.

Distributed Data Types. The distributed data type is a key and novel design in Sequoia that allows high-level interaction with decentralized data as if it were local. PPIR's distributed data types are *distributed tensor* or its low-dimension variants.

PPIR introduces a two-level data shape, where the size of the inner level corresponds to the number of data parties. As illustrated in Figure 4, $[(\texttt{50}, \texttt{50}), \texttt{784}]$ means there are 100 samples distributed over the first axis, while each of the two parties holds 50 samples. $[\texttt{100}, (\texttt{784}, \texttt{0})]$ means

```
# Grammar of PPIR
PPIR  ::=  {    lambda  <Var>*;
                let     <Eqn>*;
                in      <Var>+;
           }
Var   ::=  <name>:<data_type> | <literal>
Eqn   ::=  <operation> [<Param>*] <Var>+
Param ::=  <name>=<Var>
```

Terms	Definition
name	name of variable, enumerated locally as a, b, ..., aa, ab.
literal	constant values, e.g, a predefined threshold value.
data_type	data type and shape, see §3.1 for details
operation	PPIR-defined set of operators, see §3.2 for details.

Fig. 3. The syntax of PPIR. PPIR provides enhanced expressiveness for PPML algorithms with unified data type for distributed data and target-specific operations.

```
# Centralized dataset, with batch size of 100,
# feature size of 784, and label size of 10
x:f32[100, 784], y:i32[100, 10]

# Horizontally (sample-based) distributed definition
x:f32_d[(50, 50), 784], y:i32_d[(50, 50), 10]

# Vertically (feature-based) distributed definition
x:f32_d[100, (784, 0)], y:i32_d[100, (0, 10)]
```

Fig. 4. Example definition of distributed data types.

the data is distributed over the second axis, where the first data party possesses all 784 features, and [100, (0, 10)] means the second data party has all the labels.

To interact with distributed data types, PPIR defines PPML-related operations (§3.2) to express cross-party calculations over distributed data, such as secured matrix multiplication. However, to actually carry out the cross-party operations, they need to undergo internal transformations during the compilation of PPIR, where the operations are converted into single-party steps based on secure protocols, while remaining in the PPIR format. Details about the compilation process are in §4.

Agonostic for Intra-party Operations. PPIR does not directly manage distributed computations *within* a data party but views a single data party's computing resource as a unified entity. This design enables the executor (§5) to dynamically assign computation tasks within a party to its compute resource during runtime, giving the executor complete control over optimizing and executing intra-party computations. This approach is also necessary because a single data party's computing resource might be dynamic.

3.2 PPIR Operations

PPIR extends the standard set of ML computation operations with a list of new operations designed to support the secure computation protocols used in PPML algorithms, as shown in Table 1.

ML Computation. These operations are designed to correspond to widely adopted neural networks (NN) and ML primitives supported by lower-level computation libraries such as CuDNN. The executor of PPIR can map these operations to their computation backends.

Table 1. Types of PPIR operations supported by Sequoia.

Operations	Examples
ML Computation	*Mathematical Primitives* `mul`, `matmul`, `dot`, `add`, `sin`, `max`, ...
Secure Computation	*High-level Abstractions* `s_add`, `s_matmul`, `s_max`, `s_dot`, ... *Cryptographic Operators [12, 24]* `he_encrypt`, `ss_max`, ...
Comm. Operations	*Peer-to-Peer Communications* `send`, `recv`, `all_to_all` *Collective Communications & Computation* `all_gather`, `all_reduce`, `broadcast`, ...
Custom Operations	*Specific to PPML Algorithm [44, 45, 57]* `fedavg`, `fedsgd`, `fedyogi`, ... *Specific to Specialized Accelerator [59]* `fpga_matmul`, `rdma_read`, `square_add`, ...

Secure Computation. Cross-silo PPML utilizes security techniques such as homomorphic encryption, secret sharing and differential privacy, in this paper, they are collectively termed *secure computation*. PPIR defines two levels of secure computation:

(1) High-level abstractions are standard ML computations but over distributed data and needed to be protected, with the prefix "*s_*". During compilation (§4). They will be broken down into low-level computations based on the PPML algorithm used.

(2) Low-level computations that are specific to secure protocols like homomorphic encryption, which can be executed using standard implementation or specialized accelerators.

Communication Operations. PPML communication differs from collective communication in distributed ML because secure computation demands interleaved data exchange. When designing PPIR, we explicitly expose communication operations as *first-class citizen*, hence decoupling communication from the computation. This design allows the underlying executor to exploit overlapping opportunities between communication and computation tasks.

Custom Operations. PPIR further allows adding custom operations, such as a new secure aggregation scheme. This extensibility lets users expand the range of PPML algorithms supported by Sequoia and seamlessly connect them with Sequoia backend. Moreover, user-defined operations can be implemented tailored to the target platform, giving users access to high-performance capabilities in specialized accelerators.

Operations Supported by Sequoia. When executing PPIR (§5), Sequoia supports all basic ML computation operators through JAX [16] and also provides corresponding high-level abstractions prefixed with "s_" for secure ML computations over distributed data. These high-level abstractions undergo compilation passes (§4) and are transformed into low-level or custom operations based on specific PPML algorithms, including cryptographic operators and custom operations in PPIR.

The lower-level secure operations and custom operations implemented in Sequoia include homomorphic encryption operators, secret sharing operators, and secure aggregation operators, which are used in our evaluation (§7).

```
{
 # batch size of 100 and splitting into 2 parties
 lambda w:f32[784, 10],  b:f32[10],
 x:f32_d[(50, 50), 784], y:f32[(50, 50), 10];
 let    e:f32_d[(50, 50), 10]  = s_dot x w
 # showing only the final parameter update steps
 cl:f32_d[(50, 50), 10] = s_add bz cg
 cm:f32[784, 10]        = s_dot cl x
 cn:f32[784, 10]        = mul 0.1 cm
 co:f32[784, 10]        = sub a cn
 cp:f32[10]             = mul 0.1 ck
 cq:f32[10]             = sub b cp;
 in     co, cq;
}
```

Fig. 5. PPIR for gradient descent and parameter update in horizontal neural network training.

```
{
 # feature size of 16 and distributed across 2 parties
 lambda x:f32_d[16, (8, 8)], w:f32_d[(8, 8)], b:f32[];
 let    d:f32[16] = s_dot x w
 e:f32[16] = add d b
 f:f32[16] = sigmoid e;
 in     f;
}
```

Fig. 6. PPIR for inference in vertical logistics regression. Input variables are renamed for readability.

Communication operations supported in Sequoia are implemented through NCCL [4], including all peer-to-peer communications [6] and collective communications [5].

Details on how the compilation process utilizes low-level operations in PPML algorithms and adds support for new operators can be found in §4.4.

3.3 Examples using PPIR

This section provides examples of how PPIR expresses two common PPML paradigms: vertical (feature-based) and horizontal (sample-based). The definitions of these schemes are discussed in §2.1.

Horizontal NN. Figure 5 demonstrates the steps of gradient descent and parameter update for horizontal NN training in PPML. Similarly, the input data shape definition indicates that the data is distributed along the first axis (sample axis), as in horizontal PPML.

Vertical Logistic Regression. Figure 6 illustrates the steps involved in performing inference using vertical logistic regression in PPIR. The high-level operator **s_dot** is used for computation over distributed input data: **s_dot** takes two distributed variables as input and produces a local variable as output. The definition of input data shape ($x : \texttt{f32_d}[16, (8, 8)]$) indicates that the data is distributed along the second axis, which corresponds to the feature axis, meaning that vertical PPML should apply here.

Two-Level Operation Abstraction. Both examples provided below involve operations denoted by **s_dot**, which take distributed variables as input and produce a local variable as output. As discussed earlier in §3.2, these are high-level abstractions (prefixed with **s_**), indicating the need for cross-party computations.

To transform these high-level cross-party operations into lower-level ones that can be executed within each data party, Sequoia offers an extensible compiling mechanism where specific PPML algorithms are incorporated as transformation policies (refer to §4.3 for details and examples). The PPML algorithm used as the transformation policy is selected by the user (§4.1).

4 Sequoia API and Compiler

In this section, we introduce Sequoia's front-end component, API, and PPML-enabled compiler. The design goal is:

- Hide the model-independent secure protocols in PPML for the non-expert to program ML model with Sequoia's API.
- Transparently incorporate secure computation protocols into the ML model and generate PPIR for execution.

4.1 Programming Model

Sequoia uses JAX/NumPy-compatible API design for users to program their ML models. The user's ML code defines the function for model inference and loss calculation, while gradients are computed automatically through auto-differentiation.

In the user's program, data distribution among parties is explicitly defined using distributed data types and their shapes, enabling PPIR to manage data exchanges as discussed in §3. This definition aligns with the cross-silo PPML paradigm, which involves data distributed across multiple known parties.

An example of an ML model programmed with Sequoia's API is shown in Figure 7, and highlights the two steps needed to program the ML model and enable PPML in Sequoia:

Step 1: Sequoia provides an intuitive syntax to declare a distributed data type and map data partitions to parties, which simplifies the definition of data distribution. The cardinality of the distributed dimension is defined in the data type's shape. For external parties, cardinality is set to `None`, but the shape indicates the number of dimensions. The PPML algorithm determines whether the cardinality of an external party's data partition needs to be synchronized with others.

```
from sequoia import ppml, FEDAVG, int_d

RANK = os.environ["SEQUOIA_RANK"]

# step 1: prepare the distributed data
images_l, labels_l = load(batch_size=50)
images_d = int_d(images_l, shape=((50, None), 784))
labels_d = int_d(labels_l, shape=((50, None), 10))
params = init_network_params(layer_sizes)

# step 2: enable PPML with ML model over distributed data
def loss(params, x, y):
    preds = model.predict(params, x)
    return cross_entropy_loss(preds, y)

@ppml(party_idx = RANK, scheme = FEDAVG, args)
def update(params, x, y):
  grads = grad(loss)(params, x, y)
  return [(w - step_size * dw, b - step_size * db)
          for (w, b), (dw, db) in zip(params, grads)]

params = update(params, images_d, labels_d)
```

Fig. 7. A prototype implementation of ML models using Sequoia API, with one round of model parameter update. Some common imports and definitions are omitted for brevity.

Step 2: Users program ML models as if all computations occur locally and use the `@ppml` decorator to trigger compilation, where Sequoia incorporates PPML algorithms to perform cross-party operations.

In summary, Sequoia offers APIs that allow users to define and program their ML models as if computations occur locally. It employs compilation passes to add secure computation protocols necessary for cross-party operations in PPML algorithms.

This section will describe the two compilation passes in Sequoia that convert the user program into PPIR and explain how the transformation policies incorporate PPML algorithms. Figure 8 depicts an overview of the compiler's workflow described in this section.

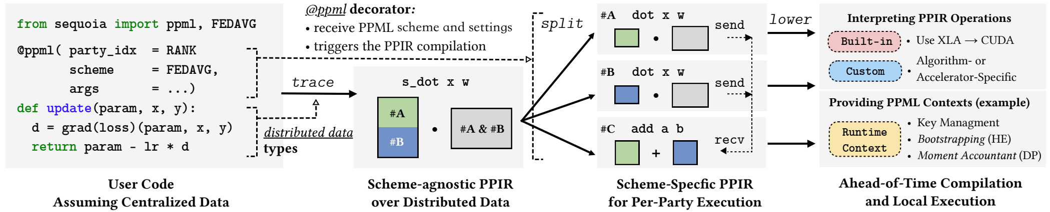


Fig. 8. Sequoia compiles user programs to PPIR in two passes. The first pass (*trace*) abstractly evaluates distributed data type and generates scheme-agnostic PPIR with high-level cross-party operation from a centralized perspective. The second pass (*split*) transforms the cross-party PPML operations into secure computations in each data party, following transformation policies defined for a specified PPML scheme. The final PPIR is *lowered* to machine code by Sequoia executor for execution.

4.2 Compilation Pass: Trace

In the first compilation pass named ***trace***, the aim is to create a centralized representation of the PPML application, identifying cross-party operations and data distributions clearly.

Decorator. Sequoia introduces a Python decorator `@ppml` to locate the compilation target while also receiving necessary configurations from the user. Sequoia traverses the syntax tree of the user code and translates it to ML and PPML operations expressed in PPIR (Figure 7, step 2).

Abstract Evaluation. Sequoia traces the user program and identifies the ML computation steps over distributed data, which will require secure computations across parties. This is statically inferred from user code by abstractly evaluating the data shape declared in user code (Figure 7, step 1).

Generating Scheme-agnostic PPIR. Sequoia translate ML computation over data distributed across parties to high-level secure computation operations in PPIR, such as `s_dot` and `s_add`, to mark them for further processing in the downstream compilation pass. The user-declared distributed data shapes are directly mapped into distributed data types in PPIR. This process is PPML-scheme-agnostic, as it generates PPIR that is not specific to any particular PPML algorithm.

The goal of this pass is to get a clear representation of PPML application from a centralized perspective, before applying any PPML algorithms. These high-level, cross-party operation abstractions in PPIR help identify which computations need to be performed securely with data distributed across parties. In the next pass, the high-level abstractions are transformed into concrete secure computations to run in each data party.

Example. In Figure 8, a simplified example shows the process between the first and second steps demonstrates this compilation pass: with the `@ppml` decorator, Sequoia finds the compilation

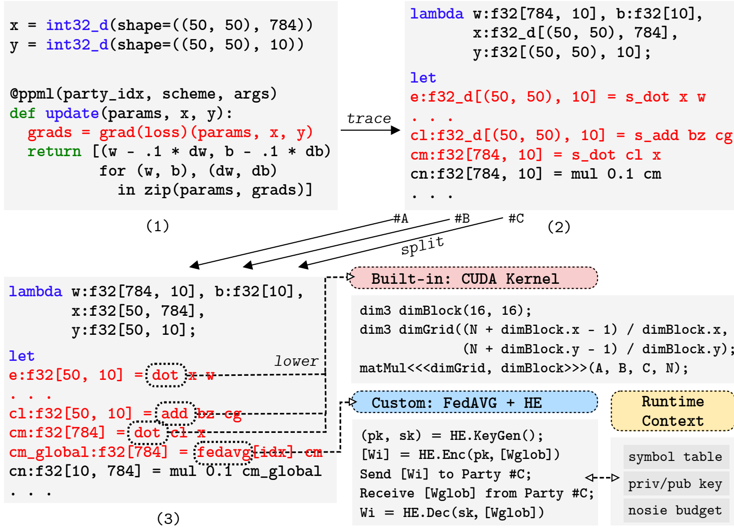


Fig. 9. PPIR compilation with FedAvg+HE: the user code specifies data distribution and PPML algorithm (FEDAVG+HE); the compiler *traces* and translates user code into PPIR with cross-party operations, and *splits* it into operations for each data party, using built-in and custom operations; runtime context stores HE information, such as keys and noise budgets.

target and translates the ML computations into high-level operation abstractions in PPIR. Sequoia automatically adds steps for calculating gradients in PPIR by applying auto differentiation (*grad* in [16]) to the original model. A more detailed example using FedAvg and HE is in Figure 9.

4.3 Compilation Pass: Split

In the second compilation pass, ***split***, Sequoia transforms the high-level cross-party operations in PPIR into concrete PPML operations to run in each data party. This is achieved by applying a transformation policy specific to an PPML algorithm.

Extensible Compilation Mechanism. In Sequoia, PPML algorithms are integrated as transformation policies, which is a set of rules applied sequentially to PPIR. In the `@ppml` decorator, it accepts a parameter named *scheme* (§4.2), which specifies the policy (i.e., the PPML algorithm) to apply.

Each rule in a transformation policy is a one-to-multiple mapping of PPIR operations, where the left-hand side (LHS) represents the original instructions in the user code, while the right-hand side (RHS) is the transformed instructions that enable PPML computation in each data party:

```
<operation> <Var, T>+ = [<Party#, operation> <Var>+,...]
```

The algorithm to apply rules is as follows:

On the LHS, the parameter T indicates the index of distributed dimension (or, axis) of the input variable: if the input variable is not distributed along the Tth axis, the rule will not match. This ensures that the transformation aligns with the expected PPML paradigms and their specific data distributions (vertical or horizontal). Sequoia will throw an error if no rules can be matched for a cross-party operation, because PPIR cannot be executed without all high-level operations being transformed.

On the RHS, the **Party#** indicates which party will locally execute this operation. PPIR's distributed data type automatically splits: when LHS is computing with distributed data, the shape of input data provided to the RHS operations corresponds to the locally available data partition.

Intuitively, the LHS of a rule is specific to high-level cross-party operations in PPIR, while the RHS is specific to the data party's computation because different data parties perform different computations depending on their roles. The RHS allows multiple entries of operations, meaning that a single operation on the LHS can be transformed into multiple operations on the RHS.

Runtime Context. PPML PPML algorithms may lose end-to-end security when broken down into their basic components, such as when intermediate results are exposed as plaintext. Therefore, the transformation policy must support cross-operator/primitive control flows when incorporating PPML algorithms, rather than only supporting individual operator transformations.

Sequoia provides a *runtime context* to share states across local operations during execution, giving PPML algorithms larger visibility of the whole program: PPML operations can write to and read from the runtime context, which can be accessed within a party by all local operations.

PPML operations can leverage runtime context to achieve data-dependent and cross-operation control flows, such as assigning symbol table attributes to variables including marking whether a variable is a cipher or plaintext, managing session private keys, monitoring the noise budget in FHE to trigger bootstrapping [12].

Lowering Scheme-Specific PPIR for Target Execution. After applying the transformation rules, the compiler generates PPIR that is specific to the PPML algorithm and data distribution. The scheme-specific PPIR will be processed by Sequoia executor (§5), leveraging existing compiler frameworks including XLA to *lower* the PPIR to machine code for execution.

Working Example. Figure 9 illustrates the compilation process for FedAvg+HE [2] (refer to the figure caption for a detailed explanation). In §4.4, we describe the specifics of adding a transformation policy with Sequoia's extensible compiling mechanism, using FedAvg+HE as an example, and covering the ruleset, runtime context utilization among other implementation details.

4.4 Extensibile Transformation Policies

In Sequoia, PPML algorithms are integrated as transformation policies, which is a set of rules applied sequentially to PPIR. In the `@ppml` decorator, it accepts a parameter named *scheme*, which specifies the policy (i.e., the PPML algorithm) to apply.

Each rule in a transformation policy is a one-to-multiple mapping of PPIR operations, where the left-hand side (LHS) represents the original instructions in the user code, while the right-hand side (RHS) is the transformed instructions to run PPML computation in *each data party*. This transformation pass, therefore, is termed *split* (§4.3).

In Figure 10, we list the transformation policy to integrate FedAvg+HE [2] to Sequoia. To improve readability, the rules are structured hierarchically, where the top part specifies the *match pattern* (conditions for applicability, e.g., operation type, variable types, party roles) and the bottom part defines the *actions* to be taken when the pattern is matched (e.g., variable encryption, aggregation operations). The bottom part may also contain nested sub-rules following the same format.

Workflow Description. The process begins with an aggregration operation (**FED_OP**), aggregating data across different parties. The FED_OP is triggered for high-level cross-party computation (s_add, s_dot) on non-cipher distributed data. For data parties, their actions include key generation, encryption, and data movements. For the aggregator, the rule generates actions to collect all the data (including from itself) and invoke the HE operand on the ciphertexts.

Note that the rule generates an OpHE operand with the first input parameter being the desired operations (*Eqn.Op*), this OpHE operand needs to be further transformed by the next rule to apply specific HE computations.

Next, the HE operation (**HE_OP**) continues the transformation to perform HE computation on aggregated data, allowing computations on encrypted data without decryption. This rule applies

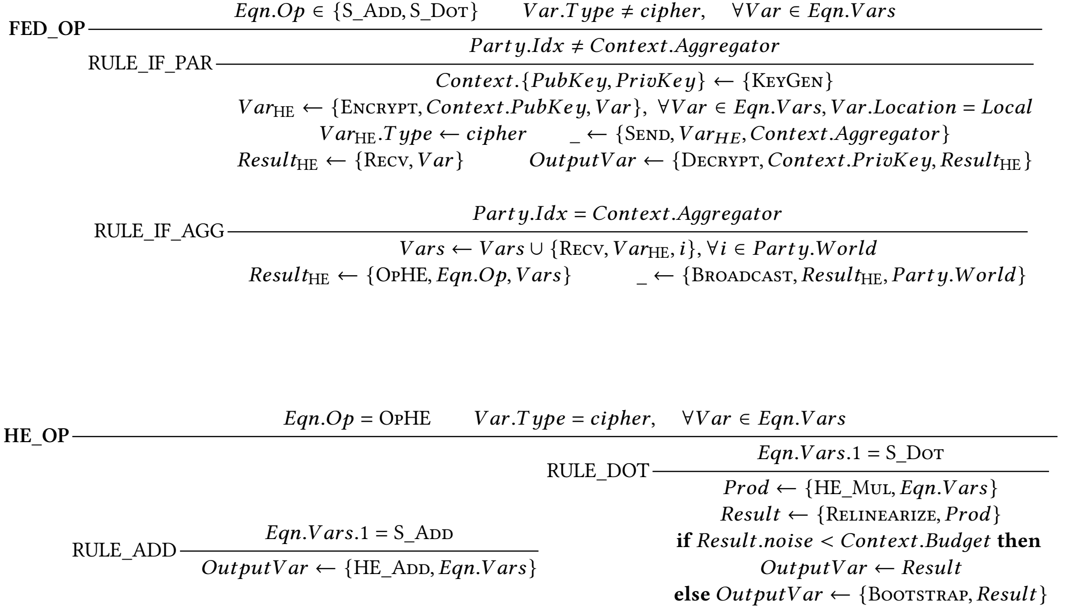


Fig. 10. The transformation policy to integrate FedAvg+HE [2]. The rules are structured hierarchically, where the top part specifies the *match pattern* (conditions for applicability) and the bottom part defines the *actions* to be taken when the pattern is matched. Nested sub-rules follow the same format. *OutputVar* is the final output.

to OpHE operand on cipher variables (which is the output of the rule above), and it performs relinearization and bootstrapping based on noise budget after multiplications.

Runtime Context. Sequoia provides a *runtime context* to share states across local operations during execution: PPML operations can write to and read from the runtime context to achieve data-dependent and cross-operation control flows.

In the example above, the runtime context is used to store:

- Variable Attributes and Symbol Table *(Var.Attr)*: Assigning attributes of variables such as encryption state (cipher or plain) and noise budget, which may affect other operations' actions. When sending variables across parties, variable attributes are transmitted together and saved to the receiver ends' runtime context.
- World Information *(Party.World, Party.Idx)*: Environment information including numbers of parties, the rank of the aggregator, and the current party's index.
- Keys *(context.PrivKey/PubKey)*: Storage of both private and public keys for HE operations. As described above, contexts are local and will not be automatically shared across parties (unless explicitly specified in PPIR by the transformation policy).

Implementation Details on Intermediate Result Transfer. Because all parties share the same scheme-agnostic PPIR before the *split* transformations (§4), the enumerated variable names (i.e., *a*, *b*,...) in PPIR are consistent across all parties. In communication operations such as *send, recv, broadcast*, the variable name itself serves as the identifier in data transfer.

Connection to Lower-Level Optimizations. Sequoia's executor (§5) lowers PPIR to machine code, where it leverages existing ML compilers like MLIR [48] and XLA [63] to generate optimized

executables on the target platform. Therefore, PPIR operator executions can be accelerated by linking specialized operator optimizations or domain-specific accelerators via custom calls in XLA, such as those for HE (e.g., HEaaN [56]) or secret sharing (e.g., BEACON [58]), when applicable. These optimizations are orthogonal to Sequoia and are discussed in §8.

Developing New PPML Algorithms. To integrate new PPML algorithms, developers need to define transformation policies for each algorithm. As previously described, these transformation policies are a set of rules that translate high-level cross-party operations into specific secure computations within each data party.

While Sequoia offers accessibility advantages for non-experts by automating the integration of secure computation protocols into ML models, expertise in secure computation is still required for adding new PPML algorithms. Developers need to understand the secure computation protocols and data distribution requirements of the algorithm to effectively define its transformation policies.

For instance, to support differential privacy (which is currently not included in Sequoia's transformation policies), Jin et al. [39] proposed a method to add noise to model parameters before sending them to the aggregator. This can be implemented by adding a new rule to insert a noise addition operation prior to invoking the Send operation for parameter aggregation.

Validating the correctness and efficiency of the PPML algorithms implemented in Sequoia is not currently considered in the system's design, and we discuss this limitation in §9.

4.5 Secure Considerations and Threat Model

Sequoia offers a generic programming abstraction for PPML applications and an extensible compilation mechanism to incorporate both existing and new PPML algorithms alongside various secure computation protocols. Its key contribution lies in optimizing the accessibility and performance of PPML applications by automating the integration of secure computation protocols into ML models.

Consequently, Sequoia is not tied to any specific PPML algorithm and does not alter the privacy-preserving properties of these algorithms (Sequoia aborts if any cross-party operation is not transformed). The threat models for PPML applications using Sequoia are inherited from the PPML algorithm selected by the user.

5 Sequoia Executor

Figure 11 illustrates the design of Sequoia's distributed execution architecture. This architecture supports a two-level hierarchy of computing resources, with per-cluster global schedulers and per-node local executors. Key features are summarized as follows:

Decentralized Cross-Party Scheduling. PPML computation involves unique data dependency patterns across data parties, and Sequoia efficiently schedules PPML computation and data movement from each data party without requiring global scheduler state synchronization. Sequoia avoids requiring a centralized system component because: (1) real-world privacy requirements may not allow deploying a centralized PPML system component. (2) the centralized component is vulnerable to network delays, especially when intermediate results are frequently exchanged in PPML.

Multi-Party, Multi-Host Execution. Real-world PPML may employ a two-level hierarchy of compute nodes, where each data party may have multiple nodes. While existing PPML frameworks often assume one node per party, Sequoia efficiently distributes PPML computations in cross- and intra-party clusters with different parallelization strategies.

5.1 Decentralized Cross-Party Scheduler

Sequoia deploys one global scheduler in each data party. Each party's global scheduler makes scheduling decisions without requiring global state synchronization from peers.

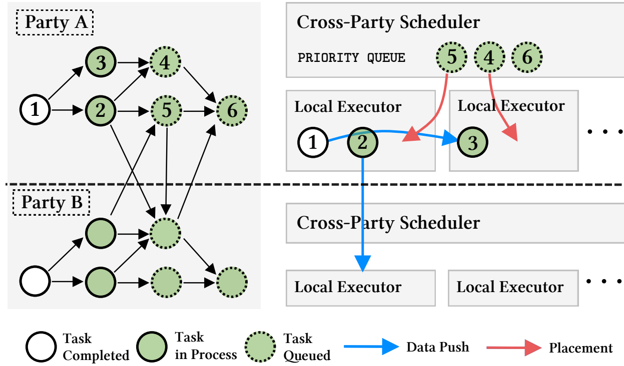


Fig. 11. Executor workflow (§5). A computation graph is constructed from PPIR, and scheduled by the cross-party schedulers. Data with dependencies are proactively transferred peer-to-peer to remote hosts, based on the early-binding placement of tasks.

Step 1: Global Kickoff. At the beginning of execution, all of the global schedulers are bootstrapped with identical copies of PPIR. The input PPIR is used to construct a global computation graph, represented as a DAG, where vertices represent tasks and edges represent dependencies (Figure 11). While the DAG includes tasks and dependencies among all parties, each party focuses on the partition of DAG relevant to their computations and communication.

Step 2: Computation Graph Consolidation. Sequoia perform vertex contractions on the computation graph, which regroups tasks at points where cross-party communication occurs. This method combines sequential local computations into one task, while still allowing cross-party parallelism and overlap. Employing larger task granularities enables better optimization, such as vectorization, by lower-level compilers [63].

Step 3: Decentralized Task Prioritization. Within each party's cluster, the global scheduler schedules local tasks and cross-party communications for external dependencies. Sequoia prioritize tasks with a decentralized scheduling strategy (i.e., no information required from other parties) by considering: (1) external dependency count: prioritizing tasks with more out-of-party successors, because cross-party communications are blocking. (2) tasks' estimated length: when dependency counts are equal, prioritize shorter tasks based on their estimated length.

Task lengths are predicted using its moving average from previous rounds, as PPML are inherently repeated in rounds, these measurements from previous rounds are sufficiently accurate [46].

Step 4: Intra-Party Early-Binding Placement. Sequoia adopts an early-binding placement strategy, making intra-party placement decisions before each training round commences: Sequoia sorts the task with priority in Step 3, and places it in the host with the shortest execution queue. Early-binding strategy offers advantages over real-time scheduling, as it enables data movements for subsequent dependent tasks to execute immediately as they are ready, instead of waiting for the location of the subsequent tasks to be determined. This allows for overlapping and asynchronous execution of computation and communication, as output data from one operation is proactively transferred to the destination host while another computation task can simultaneously be executed on that host (provided such overlapping opportunities exist).

Central Party in PPML. Sequoia employs decentralized scheduling to manage the control flow. It does *not* conflict with having a central party acting as an arbiter or aggregator. As seen in §4.3, each data party can execute different operations in PPML.

5.2 Stateless Local Executor

In cross-silo PPML, each data party may run a compute cluster. Sequoia deploys stateless local executors on each host in the cluster to carry out computations and communications.

Lowering for Local Execution. For local computation tasks, Sequoia employs ahead-of-time (AOT) compilation to execute PPIR. Specifically, XLA (which internally uses MLIR) is used to lower PPIR, generating optimized machine code for target environments. These machine codes are dynamically linked back to the Python runtime, enabling them to be cached and directly invoked in Python.

To execute secure computations, Sequoia invokes the lower-level implementations, such as SEAL [69], in existing compiling infrastructure through custom calls in XLA [48].

Specialized libraries [13, 56] and domain-specific accelerators [59, 76] can accelerate many secure computations. Since these optimizations operate in a layer lower than Sequoia (discussion in §8), their efficiency advantages can be leveraged by Sequoia through the new custom calls to their implementations, which simplifies the adoption of new secure computation backends.

Data Movement. The local executor provides a columnar format in-memory data store for tasks' input and output. To minimize communication latency, all tasks read input data from their local executor's data store, regardless of their origin. This is accomplished through *data push*, where the local executor proactively sends a task's output data to the destination host where the successor will be executed. The early placement strategy (§5.1) ensures data destinations are known before task completion. The destination can be the current host (writing to the local data store) or remote hosts within or outside the same party (network communication).

Table 2. Effectiveness of Sequoia demonstrated by LoC for accessibility and final accuracy for fidelity. NVFlare partially supports vertical PPML (non-secured), while TFF and Flower do not support vertical PPML, and are marked with ×. Baseline final accuracy is the maximum achieved among all baselines. The reduced % of LoC is over the best baseline.

PPML Setting	Model	Lines of Code (LoC)							Final Test Accuracy		
		Sequoia	Syft	TFF	NVFlare	OpenFL	Flower	TFE_{MPC}	**Sequoia**	Baselines	Centralized
Vertical	LR	10 (-92%)	127	×	×	143	×	×	81.2% (+0.0)	81.2%	81.2%
	CNN	35 (-64%)	98	×	482	120	×	×	84.2% (-0.1)	84.3%	84.8%
Horizontal	LR	10 (-64%)	112	182	102	86	130	×	80.8% (+0.1)	80.7%	81.2%
	CNN	35 (-72%)	127	230	189	151	136	144	84.8% (+0.4)	84.4%	84.8%
	ResNet18	27 (-74%)	104	217	201	147	118	×	89.8% (+0.3)	89.5%	90.3%

6 Implementation

The front-end compiler of Sequoia makes use of JAX [16], which offers a NumPy-like interface that can serve as a drop-in replacement for NumPy in ML models. JAX enables an effective toolchain for interpreting and transforming Python programs, with the Sequoia compiler extending the JAX tracer and *jaxpr* interpreter by approximately 1,600 lines of core code to support the PPML operators needed for our evaluation, including the binding for C++ libraries [69] but excluding build scripts.

The back-end executor of Sequoia consists of two components: the cross-party scheduler, which follows the logic outlined in §5.1, is implemented in native Python with about 950 lines of code; the local executor employs XLA [63] and MLIR [48] for ahead-of-time (AOT) machine code generation, as described in §5.2.

We will make the source code of Sequoia available, including the implementation of the PPML algorithms and the evaluation scripts.

7 Evaluation

We evaluate Sequoia's effectiveness and efficiency over five sets of cross-silo PPML algorithms under vertical and horizontal PPML settings, and compare them to five representation PPML frameworks. We summarize the key results as follows:

- **Effectiveness** (§7.2). Sequoia achieves similar final accuracy when compared to the PPML algorithm implemented with existing frameworks, while requiring 64%-92% fewer lines of code (LoC) to realize the same PPML computation process.
- **Efficiency** (§7.3). In the multi-party, single-host-per-party setting, Sequoia outperforms baselines by up to 88% and 252% in training throughput with up to 16 data parties for horizontal and vertical PPML, respectively. For the multi-party, multi-host setting, Sequoia achieves near-linear scalability when scaling up workers within data parties.

7.1 Methodology

Sequoia is designed to enable collaborative training of the same ML model among multiple data parties (e.g., data owners across organizations), referred to as cross-silo PPML.

Testbed Setup. We conduct our evaluation in a 32-node cluster on AWS, dividing it into up to 16 data parties. The AWS instance type used is c4.4xlarge, and for GPU experiments in §7.4, we use p3.2xlarge. To simulate the real-world network environment for cross-silo PPML, we place data parties in four AWS regions across the eastern, central, and western U.S. and connect them to the same VPN, with an average inter-region round-trip time (RTT) of 57ms.

Baselines. We select five widely adopted PPML frameworks for comparison with Sequoia, as discussed in §2.3: PySyft [9], TensorFlow Federated (TFF) [11], NVFlare by Nvidia [7], OpenFL by Intel [25], and Flower [14]. Additionally, TF Encrypted (TFE) [10] serves as an MPC-based horizontal PPML baseline. For scalability evaluations involving multiple workers in a data party, we compare Sequoia with itself using a single worker per data party, as the baselines do not support this configuration.

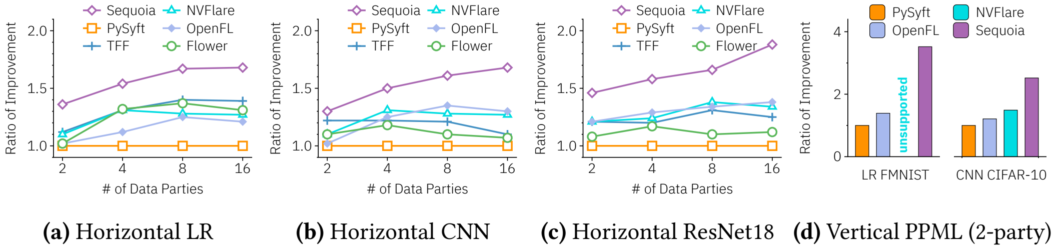
PPML Algorithms and Datasets. We evaluate Sequoia and baseline frameworks with 6 sets of PPML algorithms and datasets under vertical or horizontal cross-silo PPML settings (Table 2).

- Horizontal PPML: We use logistic regression (LR), CNN (4 layers), and ResNet18 [36] with secure aggregation through Federated Averaging (FedAvg) and Homomorphic Encryption (HE) [2, 44]. We also compare the MPC implementation of TFE and Sequoia using CNN using the semi-honest implementations of the three-party protocol ABY3 [54].
- Vertical PPML: We implement vertical logistic regression (VLR) using HE [74] to exchange intermediate results, and vertical CNN employs Split Learning [34]. [1]

In both settings, we use the Fashion-MNIST (FMNIST) dataset for the LR model and CIFAR-10 for the CNN and ResNet models. To synthesize horizontal data distribution, we randomly split the dataset into 2, 4, 8, and 16 partitions. In vertical settings, we divide the dataset into two parts: one containing all features and the other containing all labels, arranged in the same order as the features.

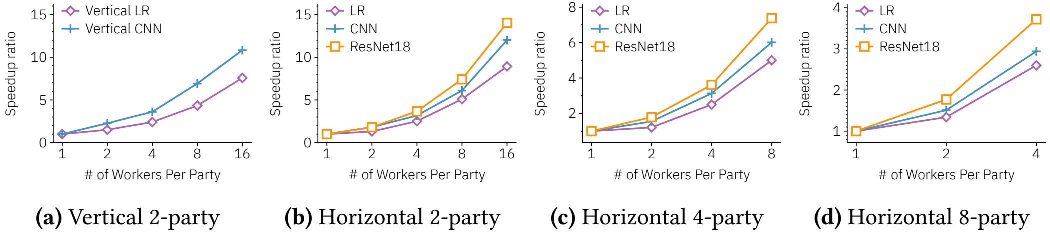
Metrics. Sequoia is evaluated by (1) the effectiveness which includes LoC (representing accessibility) and final model accuracy (representing fidelity) over different PPML algorithms. (2) its efficiency including training throughput (sample/s) performance and scalability under the multi-party, multi-host PPML setting.

[1] Vertical learning is rapidly evolving, with new algorithms continually proposed. These vertical learning algorithms are selected to demonstrate the accessibility of Sequoia.



(a) Horizontal LR (b) Horizontal CNN (c) Horizontal ResNet18 (d) Vertical PPML (2-party)

Fig. 12. Training thoughput over PPML algorithms and datasets. Sequoia outperforms baselines by up to 88% and 252% in horizontal and vertical PPML, respectively. The results are reported as improvement ratio over PySyft as it performs the worst among all baselines. Vertical PPML is only tested under 2-party setting due to the lack of multi-party vertical PPML algorithm.



(a) Vertical 2-party (b) Horizontal 2-party (c) Horizontal 4-party (d) Horizontal 8-party

Fig. 13. Scalability as the number of hosts per party grows. Results are reported by the ratio of improvement over 1 worker per party's epoch-time performance. Sequoia scales well as the number of workers grows, see analysis in §7.3.

Through deep dive experiments, we show that Sequoia's architecture can take advantage of acceleration brought by communication backends and lower-level compiler stacks, highlighting Sequoia's advantage in architectural flexibility.

7.2 Effectiveness of Sequoia

We evaluate Sequoia's effectiveness, considering its complexity for non-secure experts and its ability to maintain comparable final test accuracy in PPML-enabled ML training.

Accessibility. Sequoia is designed to be easy to use by ML researchers without requiring extensive knowledge of secure computation protocols. As shown in Table 2, PPML algorithms implemented in Sequoia take over 92% less LoC in vertical PPML and 74% less in horizontal PPML. In both the baseline and Sequoia, data preprocessing and ML model definitions with common building blocks are not counted in the LoC.

The reduction in LoC with Sequoia is primarily attributable to the reduced complexity achieved by making secured protocols transparent to users, as opposed to the baseline code which requires explicit API calls for secure computation in PPML. More specifically, the selection of the PPML algorithm (vertical, HE-based, and MPC-based horizontal) is configured with a single line of code (§4.1), alandlowing the rest of the codebase remains unchanged. As a result, the LoC for Sequoia stay consistent across these algorithms for the same model. In contrast, baseline frameworks like NVFlare require users to explicitly program control flows for each PPML algorithm. For instance, the example code [3] demonstrates that users need to utilize 15 internal APIs to set up client-server

communication, manage weight aggregation, and handle model synchronization across nodes, requiring additional learning efforts.

Despite the absolute variance in lines of code (LoC) between Sequoia and the baselines is just 100-300 LoCs, it represents significant differences in usability, such as reduced learning requirements and enhanced code comprehensibility. Since Sequoia utilizes the API from JAX, ML code—including model definitions, forward passes, and backward passes—can be directly ported to Sequoia with minimal adjustments. When developing PPML applications with Sequoia, there are two additional configurations compared to traditional centralized ML code: selecting a PPML algorithm in the @ppml decorator and defining data distribution using distributed data types. To migrate from PyTorch, this article [53] offers a comprehensive overview of the APIs for PyTorch and JAX.

Fidelity. As Sequoia transparently applied the secured computation protocol into the user-defined ML models, we evaluate the fidelity of the output program by looking at its final model accuracy. We train the same ML model under a non-PPML, centralized setting to compare. As reported in Table 2, Sequoia's fidelity is near-optimal as it achieves the same level of final model accuracy compared to non-PPML settings, while also outperforming baselines. We make a note that the differences in accuracy may be inherent in PPML algorithms, such as due to bias in cross-party data distribution and rounding differences in secured computations.

7.3 Efficiency of Sequoia

We look into Sequoia's efficiency by evaluating its training throughput and scalability.

Training Throughput. Figure 12 reports Sequoia's training throughput (samples per second). The results are reported as an improvement ratio over PySyft as it performs the worst among all frameworks. Overall, Sequoia outperforms baselines by up to 252% and 88% in vertical and horizontal PPML, respectively.

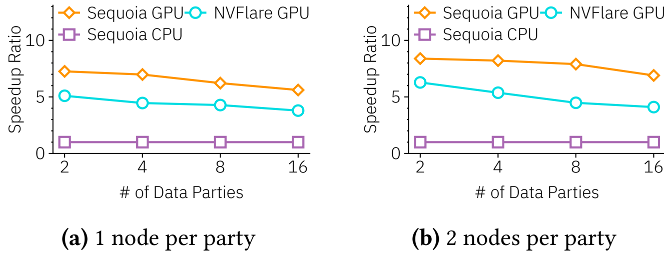
In vertical PPML, with more complex cross-parties dependencies, Sequoia's scheduler prioritizes computation tasks over others (as introduced in §5.1), to overlap computation with communication. Additionally, lower-level compiler optimization from XLA is used to maximize the degree of parallelism in the vertical LR algorithm, which also contributes to Sequoia's performance shown in related experiments.

In horizontal PPML, this improvement mostly comes from the lower system and communication overhead when synchronizing parameters across data parties. For cross-party communications, Sequoia uses optimized MPI libraries [35] for collective communication while the baseline framework uses RPC or HTTP-based transport that requires data serialization and deserialization. There is a general trend in horizontal PPML that Sequoia performs better as the number of data parties grows and the communication overhead becomes more significant.

For MPC-based horizontal ML, we evaluate the semi-honest three-party MPC (3PC) protocol [54], which is a common MPC protocol for ML. The result shows that Sequoia achieves 1.28x higher training throughput than TFE by overlapping computation and communication. Since the MPC protocol does not support scaling, it was not included in scalability experiments.

Intra-party Scalability. Figure 13 shows that Sequoia accelerates intra-party computation by scaling up workers inside the data parties. The compute nodes within a data party work in the data-parallel mode and synchronize gradients at the end of each iteration before sharing with other data parties. Overall, Sequoia achieves near-linear scalability for the intra-party computation.

In vertical PPML, Sequoia achieves 10.83× speedup in the 16-worker (per party) setting. The intra-party system synchronization overhead is larger than the overhead in the horizontal setting, because vertical PPML shares the intermediate results more frequently the horizontal PPML.



(a) 1 node per party **(b)** 2 nodes per party

Fig. 14. PPIR facilitates integration with existing lower-level compilers. By switching to GPU as the computation backend, Sequoia achieves 839% speedup over CPU, while also outperforming NVFlare on GPU by up to 68%.

In horizontal PPML, we evaluate Sequoia under 2, 4, and 8-party settings, with a maximum of 32 workers in total. Sequoia slightly degrades in LR and CNN, as they have fewer model parameters, making the optimization for communication less significant.

Summary. The efficiency experiment results demonstrate the advantages of Sequoia workflow compared to baselines, which significantly reduces cross-party coordination overhead for cross-silo PPML. Furthermore, although Sequoia is not explicitly designed to optimize specific PPML algorithms, introducing high-level abstractions allows for the integration of ML system optimization techniques, such as scheduling, and optimizes the execution of PPML by a considerable margin.

7.4 GPU Experiments

As previously discussed in §3, the use of PPIR and a compiler-executor architecture allows for easy integration with existing lower-level compilers, enabling the generation of optimized executables on the target platform. Sequoia has the potential to improve efficiency even further by leveraging specialized hardware accelerators for PPML operations.

To show the extensibility of Sequoia's compiler-executor architecture, we switch out the target platform in XLA to use GPU as the computation backend, and switch to MPI4JAX [35] as the communication backend in the horizontal CNN experiment. As shown in Figure 14, Sequoia achieves up to 839% speedup over CPU and the scalability downgrades gracefully, while also outperforming NVFlare on GPU by up to 68%. NVFlare's centralized, job-submission workflow leads to more significant synchronization overhead, as using GPU for computation increases the relative ratio of communication.

8 Related Work and Discussions

Prior PPML frameworks are discussed in §2. In this section, we discuss other optimizations for privacy-preserving computations and their relations to Sequoia.

Optimizations for Homomorphic Encryption (HE) and Multi-Party Computation (MPC). HE and MPC compilers mainly focus on two optimization directions:

1. *Computation optimizations* such as operator fusion and auto parallelization for specific operators [17, 19, 31, 56, 58]. For example, HEaaN.MLIR [56] optimizes specific HE operators with more efficient arithmetic algorithms and automated parallelization. These optimizations operate at a layer below Sequoia's compiler and can be seamlessly linked to Sequoia, as discussed in §5.2. For example, HEaaN.MLIR [56] is part of the MLIR compiler infrastructure used by Sequoia to execute PPIR.

2. *New programming abstractions* that transparently uses existing secure computation libraries [20, 22, 52, 70]. For example, EVA [22] introduces a high-level FHE language and performs operator-level reorganization that automatically inserts relinearization and rescaling operations to generate

correct FHE programs. SecretFlow [52] transforms the user's ML model, but its transformation capability limits to operator transformation in ML and MPC, limiting its adaptability to other complex workflows in PPML algorithm. Sequoia's compiler uses abstract evaluation to achieve transparent and extensible transformations, avoiding adding another layer of abstraction.

Domain Specific Accelerators for Privacy-Preserving Computations. DSA is an emerging research area that explores hardware accelerators, such as FPGAs and ASICs, to boost the performance of secure computation workloads, which suffer from over a 60× slowdown [76] in end-to-end performance compared to plaintext computation [59, 66, 67, 76]. HEAX [59] is a hardware architecture that accelerates number-theoretic transform, a fundamental building block for FHE, with multiple levels of parallelism. FLASH [76] is an FPGA hardware accelerator for cross-silo PPML. It optimizes nine cryptographic operations (including HE and RSA) which are widely used in cross-silo PPML algorithms.

Sequoia facilitates the integration of accelerators into PPML frameworks without altering user applications. In §5.2, we explain that Sequoia's executor converts PPIR to machine code for execution on the target platform. As custom PPIR operations have lower-level implementations, hardware accelerators can be called for these operations when possible.

9 Limitations

Implementation and Validation of PPML Algorithm. The Sequoia compiler is designed to be extensible, allowing users to add new PPML algorithms as transformation policies. However, we acknowledge that implementing new PPML algorithms requires specialized expertise in secure computation protocols and PPML techniques. Furthermore, validating the correctness and efficiency of these algorithms also depends on the user's technical expertise. It is an interesting future work to explore methods that automate the generation and validation of transformation policies for new PPML algorithms.

Support of Large Models. Newer, larger models with complex architectures, incorporating advanced training paradigms and optimizations (e.g., FSDP [77] and FlashAttention [21]), require specialized GPU operations and sophisticated computation and communication scheduling. While basic transformer models can run on JAX and therefore on Sequoia, their most efficient implementations may not be directly compatible with Sequoia without additional development. This limitation arises from two primary factors:

ML operators that rely on custom CUDA kernels and enforce more refined GPU memory management are not fully supported by JAX, and therefore not by Sequoia. Advanced parallelization strategies require manual implementation with newer APIs in JAX, which may not be fully compatible with Sequoia's compiler. It is also worth noting that large models are often pretrained on substantial amounts of public data, and recent research has addressed privacy concerns for these models from perspectives beyond distributed data ownership, such as prompt security and personal data leakage. These issues are distinct from the distributed data scenarios addressed in this paper and are typically handled at the ML algorithm level, outside of frameworks like PyTorch, JAX, or Sequoia.

10 Conclusion

This paper proposes Sequoia, a novel PPML framework that decouples ML models and secure protocols. Sequoia offers JAX-compatible APIs for users to program their ML models, and automatically apply PPML algorithms for execution over distributed data to improve the accessibility and performance of PPML applications.

Acknowledgments

We thank the anonymous reviewers from SIGMOD 2025, as well as reviewers from our previous submissions, for their valuable feedback. We thank Xingxing Tang for his help and discussion in the early stage of this project.

This work is supported in part by the Hong Kong RGC TRS T41-603/20R, GRF 16213621, NSFC 62062005, 62402407 and the Turing AI Computing Cloud (TACC) [72]. Kai Chen is the corresponding author.

References

[1] FATE. https://github.com/FederatedAI/FATE.
[2] Federated Learning with Homomorphic Encryption. https://developer.nvidia.com/blog/federated-learning-with-homomorphic-encryption. Nvidia.
[3] Hello PyTorch - Nvidia FLARE 2.4.0 documentation. https://nvflare.readthedocs.io/en/2.4/examples/hello_pt.html.
[4] NCCL. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/overview.html.
[5] NCCL collective operations. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html.
[6] NCCL point-to-point communication. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/p2p.html.
[7] NVFlare by Nvidia. https://github.com/NVIDIA/NVFlare.
[8] Nvidia NVFlare PyTorch Example. https://nvflare.readthedocs.io/en/main/examples/hello_pt.html.
[9] Pysyft by openmined. https://github.com/OpenMined/PySyft.
[10] Tensorflow Encrypted. https://github.com/tf-encrypted/tf-encrypted.
[11] Tensorflow Federated. https://www.tensorflow.org/federated.
[12] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.
[13] Shashank Balla and Farinaz Koushanfar. Heliks: He linear algebra kernels for secure inference. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 2306–2320, New York, NY, USA, 2023. Association for Computing Machinery.
[14] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmao, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2020.
[15] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.
[16] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
[17] Di Chai, Junxue Zhang, Liu Yang, Yilun Jin, Leye Wang, Kai Chen, and Qiang Yang. Efficient decentralized federated singular vector decomposition. In *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10-12, 2024*, pages 1029–1047. USENIX Association, 2024.
[18] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 36(6):87–98, 2021.
[19] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2017.
[20] Sangeeta Chowdhary, Wei Dai, Kim Laine, and Olli Saarikivi. Eva improved: Compiler and extension library for ckks. *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2021.
[21] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher R'e. Flashattention: Fast and memory-efficient exact attention with io-awareness. *ArXiv*, abs/2205.14135, 2022.
[22] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. Eva: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2020.
[23] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
[24] Wenliang Du and Mikhail J Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms*, pages 13–22, 2001.
[25] Patrick Foley, Micah J Sheller, Brandon Edwards, Sarthak Pati, Walter Riviera, Mansi Sharma, Prakash Narayana Moorthy, Shih han Wang, Jason Martin, Parsa Mirhaji, Prashant Shah, and Spyridon Bakas. OpenFL: the open federated

learning library.
[26] Lars Folkerts, Charles Gouert, and Nektarios Georgios Tsoutsos. Redsec: Running encrypted discretized neural networks in seconds. *Proceedings 2023 Network and Distributed System Security Symposium*, 2023.
[27] David Froelicher, Hyunghoon Cho, Manaswitha Edupalli, Joao Sa Sousa, Jean-Philippe Bossuat, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, Bonnie Berger, and Jean-Pierre Hubaux. Scalable and privacy-preserving federated principal component analysis. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1908–1925, 2023.
[28] Fangcheng Fu, Yingxia Shao, Lele Yu, Jiawei Jiang, Huanran Xue, Yangyu Tao, and Bin Cui. VF2Boost: Very fast vertical federated gradient boosting for cross-enterprise learning. In *Proceedings of the 2021 International Conference on Management of Data*, pages 563–576, 2021.
[29] Shaoduo Gan, Xiangru Lian, Rui Wang, Jianbin Chang, Chengjun Liu, Hongmei Shi, Shengzhuo Zhang, Xianghong Li, Tengxu Sun, Jiawei Jiang, Binhang Yuan, Sen Yang, Ji Liu, and Ce Zhang. BAGUA: scaling up distributed learning with system relaxations. *Proc. VLDB Endow.*, 15(4):804–813, 2021.
[30] Michelle Goddard. The eu general data protection regulation (gdpr): European regulation that has a global impact. *International Journal of Market Research*, 2017.
[31] Sanath Govindarajan and William S. Moses. Syfer-mlir: Integrating fully homomorphic encryption into the mlir compiler framework. 2020.
[32] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. GRAPHENE: Packing and Dependency-Aware scheduling for Data-Parallel clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 81–97, Savannah, GA, November 2016. USENIX Association.
[33] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. In *Privacy Enhancing technologies Symposium (PETS) 2024*, June 2024.
[34] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.
[35] Dion Hafner and Filippo Vicentini. mpi4jax: Zero-copy mpi communication of jax arrays. *Journal of Open Source Software*, 6(65):3419, 2021.
[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning. *Image Recognition*, 7, 2015.
[37] Mikko A Heikkilä, Antti Koskela, Kana Shimizu, Samuel Kaski, and Antti Honkela. Differentially private cross-silo federated learning. *arXiv preprint arXiv:2007.05553*, 2020.
[38] Xinyang Huang, Junxue Zhang, Xiaodian Cheng, Hong Zhang, Yilun Jin, Shuihai Hu, Han Tian, and Kai Chen. Accelerating privacy-preserving machine learning with genibatch. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*, pages 489–504. ACM, 2024.
[39] Weizhao Jin, Yuhang Yao, Shanshan Han, Jiajun Gu, Carlee Joe-Wong, Srivatsan Ravi, Salman Avestimehr, and Chaoyang He. Fedml-he: An efficient homomorphic-encryption-based privacy-preserving federated learning system, 2024.
[40] Georgios Kaissis. End-to-end privacy preserving deep learning on multi-institutional medical imaging. *Nature Machine Intelligence*, 2021.
[41] Pallika Kanani, Virendra J Marathe, Daniel Peterson, Rave Harpaz, and Steve Bright. Private cross-silo federated learning for extracting vaccine adverse event mentions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 490–505. Springer, 2021.
[42] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.
[43] Hiroaki Kikuchi, Chika Hamanaga, Hideo Yasunaga, Hiroki Matsui, Hideki Hashimoto, and Chun-I Fan. Privacy-preserving multiple linear regression of vertically partitioned real medical datasets. *Journal of Information Processing*, 26:638–647, 2018.
[44] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated Learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.
[45] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.
[46] Fan Lai, Jie You, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Sol: Fast distributed computation over slow networks. In *NSDI*, 2020.
[47] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 19–35. USENIX Association, July 2021.
[48] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: A compiler infrastructure for the end of moore's law. *arXiv preprint arXiv:2002.11054*, 2020.

[49] Qinbin Li, Zeyi Wen, and Bingsheng He. Practical federated gradient boosting decision trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.
[50] Zengpeng Li, Vishal Sharma, and Saraju P Mohanty. Preserving data privacy via federated learning: Challenges and solutions. *IEEE Consumer Electronics Magazine*, 9(3):8–16, 2020.
[51] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Robustness of secure federated learning. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 453–476, 2023.
[52] Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 17–33, Boston, MA, July 2023. USENIX Association.
[53] Sabrina J. Mielke. From pytorch to jax: towards neural net frameworks that purify stateful code, Mar 2020.
[54] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.
[55] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 1–15, New York, NY, USA, 2019. Association for Computing Machinery.
[56] Sunjae Park, Woosung Song, Seunghyeon Nam, Hyeongyu Kim, Junbum Shin, and Juneyoung Lee. Heaan.mlir: An optimizing compiler for fast ring-based homomorphic encryption. *Proc. ACM Program. Lang.*, 2023.
[57] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445*, 2019.
[58] Deevashwer Rathee, Anwesh Bhattacharya, Divya Gupta, Rahul Sharma, and Dawn Song. Secure floating-point training. Cryptology ePrint Archive, Paper 2023/467, 2023. https://eprint.iacr.org/2023/467.
[59] M Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. Heax: An architecture for computing on encrypted data. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1295–1309, 2020.
[60] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. Eiffel: Ensuring integrity for federated learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2535–2549, New York, NY, USA, 2022. Association for Computing Machinery.
[61] W. Ruan, M. Xu, W. Fang, L. Wang, L. Wang, and W. Han. Private, efficient, and accurate: Protecting models trained by multi-party learning with differential privacy. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1926–1943, Los Alamitos, CA, USA, may 2023. IEEE Computer Society.
[62] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning, 2018.
[63] Amit Sabne. Xla: Compiling machine learning for peak performance, 2020.
[64] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In *ACM Conference on LISP and Functional Programming*, 1992.
[65] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 3:3, 2018.
[66] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21. Association for Computing Machinery, 2021.
[67] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. Craterlake: A hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22. Association for Computing Machinery, 2022.
[68] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, João Sá Sousa, and Jean-Pierre Hubaux. Poseidon: Privacy-preserving federated neural network learning. *ArXiv*, abs/2009.00349, 2020.
[69] Microsoft SEAL (release 3.0). http://sealcrypto.org, October 2018. Microsoft Research, Redmond, WA.
[70] Tim van Elsloo, Giorgio Patrini, and Hamish Ivey-Law. Sealion: a framework for neural network inference on encrypted data, 2019.
[71] Hao Wang, Han Tian, Jingrong Chen, Xinchen Wan, Jiacheng Xia, Gaoxiong Zeng, Wei Bai, Junchen Jiang, Yong Wang, and Kai Chen. Towards domain-specific network transport for distributed DNN training. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*. USENIX Association, 2024.
[72] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. TACC: A full-stack cloud computing infrastructure for machine learning tasks. *CoRR*, abs/2110.01556, 2021.

[73] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. Federated learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13(3):1–207, 2019.

[74] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu. Parallel distributed logistic regression for vertical federated learning without third-party coordinator, 2019.

[75] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 493–506. USENIX Association, 2020.

[76] Junxue Zhang, Xiaodian Cheng, Wei Wang, Liu Yang, Jinbin Hu, and Kai Chen. FLASH: Towards a high-performance hardware acceleration architecture for cross-silo federated learning. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA, April 2023. USENIX Association.

[77] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023.

Received July 2024; revised September 2024; accepted November 2024