Rosetta: A Privacy-Preserving Computation Framework for Artificial Intelligence

Version 0.1

LatticeX Foundation

July 29, 2020

Contents

1 Introduction

Data is a new generation of production factors after land, energy, population, and food. The value of data would definitely lie in the flow of it. The essential difference between data and other factors of production is that data privacy is the fundamental starting point for generating value in the process of data flow.

At present, there is a striking contradiction between the business models of Internet giants and data privacy. The essence of the existing business model of these companies is to collect and utilize users' data to monetize. We, in the age of the internet era, have already been reduced as data carriers and tools. In this new digital era, people should and must take back the natural rights as individuals and "descend" Internet platforms into tools for human being.

Traditional Internet giants will be limited by their inherent business models, and it will be difficult to fight left and right. Each generation of business giants has its own lifecycle, and traditional Internet giants will have an uncrossable crisis in the era of privacy-preserving computation. Privacy-preserving computation is sure to be an important underpinning of the public infrastructure of the future all-digital age.

Privacy-Preserving Artificial Intelligence is the heart of privacy-preserving computation and is the foundation for how humanity will survive in the all-digital age. It promotes to reach the agreement for the benefits of all involved parties, constructs a new interaction paradigm, provides a complete and decentralized privacy-preserving solution. The balance between computational complexity and communication complexity that perfectly underpins all future human business practices in the digital economy. This is LatticeX Foundation's understanding and true value of privacy-preserving compuation.

To support privacy-preserving AI, a new architecture combining privacy-preserving computation with AI technology is urgently needed, which can make it possible to hold up upper-layer products and applications. This is the basic starting point of the Rosetta.

2 Overview of Rosetta

Artificial Intelligence technologies and algorithms are applied to every data-related scenarios. Machine learning and deep learning obtain a model by collecting a large amount of data in the process of training and inference. These procedures invovles in sensitive data, while the model itself is also very valuable.

Due to the basic model of AI technology, a major bottleneck is currently encountered, which is the issue of data privacy protection. With the amount of data exploding today, it's hard to have one organization collecting all of it. Data, as an extremely special asset, is also not really shared between businesses. The diversity of the data directly determines the accuracy of the model and thus the business costs and risks. From an AI algorithm perspective, the more data the better; from a privacy protection perspective, the less data exposed the better. This inherent "contradiction" has become the biggest obstacle to the further use of AI technology.

Privacy-preserving computation is the fundamental solution to the above "contradiction". It guarantees data privacy in the premise of AI model training and inference. Maximize the use of data diversity and also protect data privacy. Combining blockchain as the infrastructure of the distributed economy leads to a complete public infrastructure for the all-digital age.

Rosetta a privacy-preserving framework for AI, connecting AI technology with privacy-preserving computation and organically combining blockchain smart contracts. The Rosetta Stone is widely known to contain three ancient texts: ancient Egyptian hieroglyphics, Egyptian cursive, and ancient Greek. Rosetta thus symbolically carries and combines the three typical of technologies

(privacy-preserving computation, AI and blockchain) to ultimately support a complete privacy-preserving AI solution.

Privacy-Preserving Computation (PPC) technologies can be broadly divided into two categories: algorithm-based (Trustless Computation) and hardware-based (Trusted Computation).

Trustless Computation is mainly from cryptography technologies, including secure multi-party computation (MPC, [16]), zero-knowledge proof (ZKP,[7]) and fully homomorphic encryption (FHE, [13]). Federated Learning[9] solves the privacy problems from the perspective of AI algorithms, and it is inevitable to use various cryptographic technologies. It can also be categorized as the algorithm-based privacy-preserving computation.

Hardware-based privacy-preserving computation techniques, on the other hand, use hardware to create a Trusted Execution Environment (TEE, [14]) in which data can be used and computed securely. Typical TEEs are Intel's SGX [6] and ARM's TrustZone [2], and the open source framework KeyStone [11], among others.

Rosetta's ultimate goal is to support the delivery of deployable, operational, serviceable commercial products. Thus, the technical path taken is to combine commonly used AI frameworks, starting with cryptographic algorithms and gradually extending to other privacy computing techniques.The structure is described in the red dotted box in Figure1.

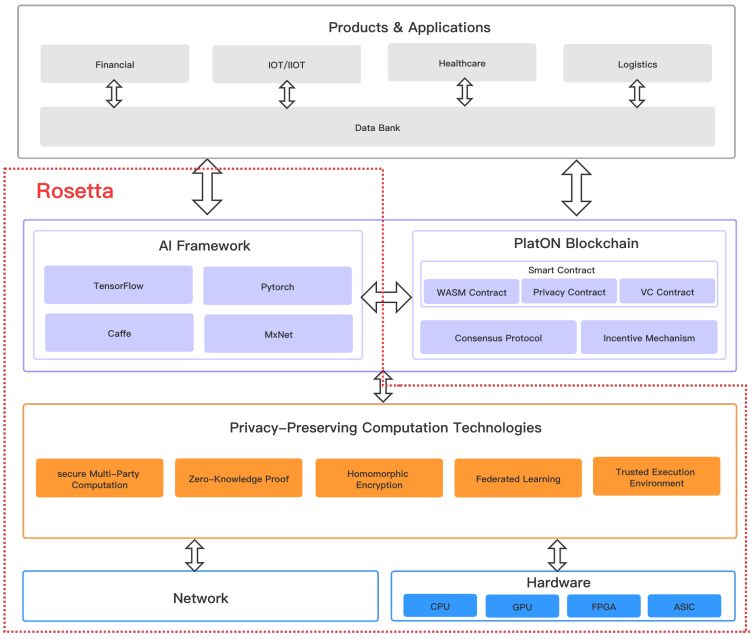

Figure 1: Overview of Rosetta

- **Network & Hardware:** The scenarios of privacy-preserving AI should be distributed, it must be supported by an underlying network. Most of the privacy-preserving computation technologies require high computational complexity and communication complexity, while solving the network communication, it also needs to be able to improve the overall operating performance through hardware acceleration. Rosetta is now CPU-based, and ultimately supports GPU, FPGA and ASIC and other high-performance hardware acceleration devices.

- **Privacy-Preserving Computation Technologies:** Rosetta will support a variety of privacy-preserving computation technologies. Including cryptography, (MPC, ZKP and FHE), Federated learning and optimized with cryptographic algorithms and TEEs.
- **AI Framework:** In order to allow AI engineers to easily use this framework, Rosetta would gradually integrate mainstream framework, consisting of TensorFlow [1], Pytorch [12], Caffe [8], Mxnet[4]. We now focus on TensorFlow.
- **Products & Applications:** Rosetta ultimately underpins the products of privacy-preserving AI and serves applications across industries, and will gradually serve the financial, IoT, healthcare and logistics industries.

2.1 Relationship with Blockchain

Privacy-Preserving Network and Blockchain are extremely closely linked as two important components of the infrastructure of the digital world. Privacy-preserving computation has the ultimate goal of achieving data assetization, which is based on data pricing. Clearly, traditional pricing based on the size of the data volume is no longer realistic or accurate. The real value of data is not in the amount of data, and we believe that the value of data must be reflected in the utility of the data, and that the data that is actually used is valuable.

A data exchange, transaction network based on privacy-preserving computation must be sufficiently data-liquid, and necessarily distributed, in full alignment with the distributed architecture of blockchain. And the initial motivation to contribute privacy data to participate in collaborative computing (without compromising privacy) needs to be underpinned by an incentive mechanism, which is the fundamental reason why blockchain mechanisms can operate steadily. Ultimately, the clearing of data assets must also be cross-enterprise and cross-industry, and only blockchain, as the infrastructure of the distributed economy, can serve as an essential tool for clearing.

In a distributed, multi-centric environment, the most effective tool for building trust is blockchain. The blockchain's consensus algorithm reaches a final agreement on the state of the data, the process by which the data will operate. The smart contract carries all kinds of privacy-preserving applications, and eventually forms a complete privacy data trading market.

Therefore, only the combination of blockchain networks and privacy-preserving networks can truly underpin the full process of data assetization.

2.2 Why Rosetta

Rosetta is a open-source tool for AI developers and researchers. Similarly, developers and researchers who understand and are familiar with privacy-preserving computation technologies are needed to support and refine them. Therefore, Rosetta's architecture is fully decoupled for these two types of potential users, giving full play to the strengths of their respective fields, while organically combining to form a complete whole, in order to achieve rapid iteration and optimization purposes. Thus, Rosetta has three important features.

- **Easy of Use:** There is a large gap between cryptography and AI disciplines. Cryptography, in particular, involves more complex, abstract mathematical tools with a high learning curve. So having AI engineers fully familiarize and master complex cryptographic algorithms can result in a very high threshold for use. AI algorithms, on the other hand, are complex and varied, and cannot be customized on a case-by-case basis through cryptographic algorithms. This is detrimental both to developer use and to subsequent extension.

Based on this, Rosetta integrates common AI frameworks. The current stage is dominated by TensorFlow, which has the largest market share. Privacy-preserving computation algorithms are used to transform the various AI operations in the framework into privacy operations, i.e., functionally invariant, with data privacy protection. This allows the AI engineer/researcher to piece together the desired model using the underlying privacy arithmetic in the way traditional TensorFlow is used.

In order to further lower the threshold for developers, Rosetta has retained the TensorFlow interface API, which allows AI developers to use TensorFlow to adjust the model with plain data, and once multiple data is introduced for processing, simply introducing the following package can be automatically converted to a privacy-preserving version.

```
import latticex.rosetta
```

- **High Efficiency:** Privacy-preserving computation algorithms have extremely high performance requirements. Cryptography algorithms, for example, are more likely to be implemented using efficient programming languages such as C/C++ due to the unique mathematical modules they use. Although the underlying implementation of TensorFlow also uses C/C++, the purposes of the two are fundamentally different. Cryptography requires a faster library of tools for algebraic operations, while the underlying implementation of TensorFlow is more optimized for numerical operations. Therefore, using TensorFlow directly to implement the cryptographic algorithm will have more limitations and will also bring certain performance bottlenecks. Also, since most cryptographic algorithms are based on C/C++ implementations, moving them to TensorFlow implementations is bound to result in a huge amount of algorithm refactoring.

  On the other hand, as an industrial grade product, TensorFlow is optimized on many levels for industrial use. For example, the splitting of subdiagrams, parallel execution of diagrams, etc., are all fully reusable in the execution of the privacy operations.

  Based on this, Rosetta still maintains the high performance of the underlying C/C++ implementation of the cryptographic algorithms, while organically combining the TensorFlow framework's industrial-grade optimization for parallel computing to ensure the high efficiency of the algorithm while maintaining ease of use.

- **High Flexibility:** For the same privacy operations, multiple implementations of cryptographic algorithms exist. Each algorithm has unique advantages that apply to different scenarios. Even the same scenario has different cryptographic algorithms to match different deployment environments.

  Based on this, Rosetta will flexibly support a variety of privacy-preserving computation algorithms, including secure multiparty computation (MPC), zero-knowledge proof (ZKP) and homomorphic encryption (HE), as well as federated learning (FL) and trusted execution environment (TEE). In a secure multi-party computation implementation, for example, the cryptographic algorithm will first integrate various 3-party protocols and gradually expand to N-party protocols. The implementation of cryptographic algorithms can be completely independent of the framework, which provides sufficient flexibility for cryptographic developers. At the same time, Rosetta provides a rich interface for easy integration of the algorithm with the framework.

2.3 Roadmap

In line with the maturity of technologies and requirements, Rosetta has developed the following roadmap in Figure 2:

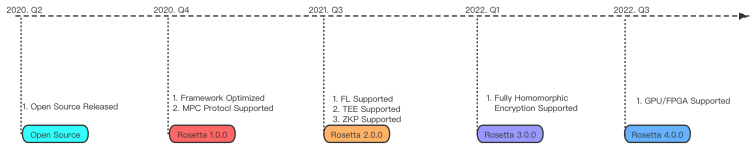

Figure 2:

- **Open Source (2020Q2)** Rosetta's overall architecture will be officially open source in late April/early May 2020. This version completes the basic framework design for integration with TensorFlow, integrating secure multi-party computing protocols that enable private training and inference of simple machine learning algorithms.
- **Rosetta v1.0.0 (2020Q4)** Rosetta version 1.0.0 will be officially released in the fourth quarter of 2020. This release completes the optimization of the architecture and algorithms and supports the conversion of all operations in TensorFlow to privacy operations.
- **Rosetta v2.0.0 (2021Q3)** Rosetta version 2.0.0 will be released in the third quarter of 2021. Other privacy preserving algorithms, including federal learning and trusted execution environments, are implemented and integrated. Zero-knowledge proof algorithms for cryptography will also be supported.
- **Rosetta v3.0.0 (2022Q1)** Rosetta version 3.0.0 will be released in the first quarter of 2022 and the main feature will be the integration of full homomorphic encryption algorithms. The CKKS [5] scheme suitable for numerical computation will be used in this version for the numerical computation requirements of AI algorithms.
- **Rosetta v4.0.0 (2022Q3)** Rosetta version 4.0.0 is a relatively complete hardware and software combination that will support and integrate various hardware acceleration schemes to reduce overall computing complexity.

3 Architecture

In this section, the details of Rosetta's architecture and principles will be introduced. To better understand the initial design of Rosetta, we will introduce the architecture of TensorFlow and the principles of privacy-preserving compution algorithms. Secure multiparty computing is introduced as an example.

3.1 Overview of TensorFlow

TensorFlow uses Data Flow Graphs for numerical calculations, as shown in Figure 3. In the diagram, nodes represent mathematical operations such as matrix addition, matrix multiplication, etc. The lines (Edges) in the figure represent an array of interconnected multidimensional data, called tensor, between nodes. Nodes and lines form a Directed Graph, with the direction

of the lines indicating the input and output of the data and the tensor flowing through the graph indicating the operation performed on the data, which is where the TensorFlow name comes from.

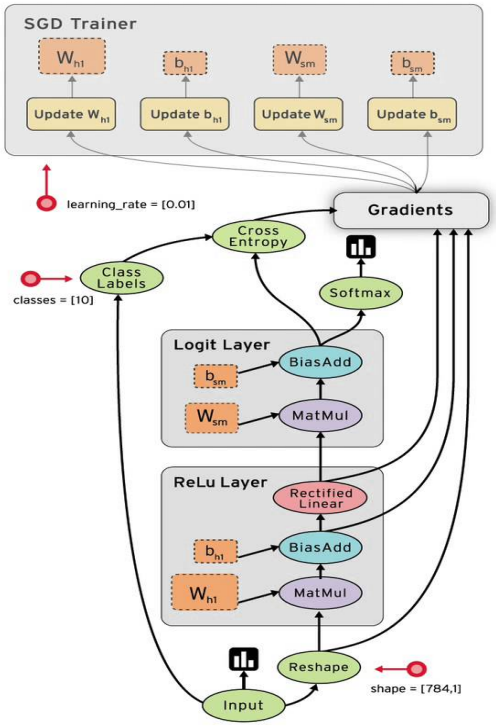

Figure 3: Data Flow Graph (http://www.tensorfly.cn/)

TensorFlow separates the definition of a computation from its execution. The frontend allows developers to define the computations (such as an AI model) that need to be performed using a low-threshold programming language (Python), which TensorFlow automatically converts into a directed graph. The following code block describes the entire computation process in Python, and the Figure 4 shows the autotransformed directed graph.

```
import tensorflow as tf

b = tf.Variable(tf.zeros([128]), name="b")
W = tf.Variable(tf.random_uniform([784 ,1],-1,1), name="w")
x = tf.Variable(tf.random_uniform([128 , 784]), name="x")
relu = tf.nn.relu(tf.matmul(x, W) + b)
init = tf.global_variables_initializer ()
with tf.Session() as sess:
  sess.run(init)
  print(sess.run(relu))
Writer = tf.summary.FileWriter("log", tf.get_default_graph())
Writer.close()
```

The node of the directed graph represents an operation, which is implemented in the TensorFlow Kernel in C/C++. After a given input data, the backend performs the entire computation

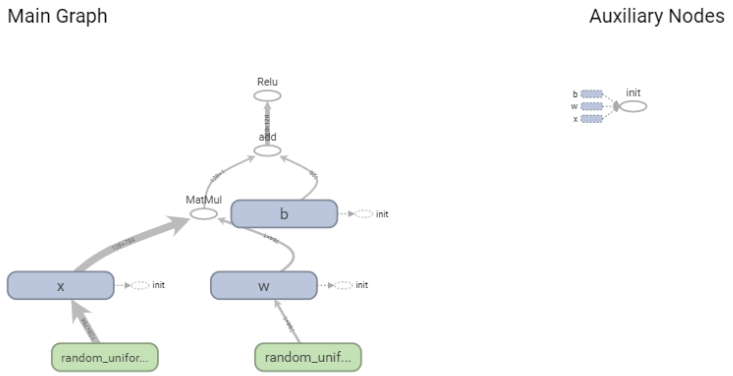

Figure 4: Directed Graph Generated with TensorBoard

according to the structure of a directed graph. TensoFlow makes a number of industrial-grade optimizations in the generation of graph and the execution of calculations, including graph/sub-graph structure, parallel execution, and other optimizations. Rosetta will leverage TensorFlow's existing industrial-grade optimizations wherever possible.

3.2 Overview of Multi-Party Computation

Secure Multi-Party Computation [16] was originally proposed by Andrew Chichi Yao to compute functions between multiple participants without compromising input privacy. There are many protocols that implement secure multi-party computation, and here we mainly present the algorithm based on Secret Sharing.

For simplicity, we consider the situation that contains two participants P_1 and P_2, with another helper P_0 for random number generation. The semi-honest adversarial model is the only one we consider here. Furthermore, we only consider the honest-majority model in this article, which means no more than two nodes would collude among these three parties. This article considers `uint64_t` data type.

Given a number x, define $\mathsf{share}(x) = (x - r, r)$, where x is a uniformly random number. One could see that if $\mathsf{share}(x) = (x_1, x_2)$, then $x = x_1 + x_2$, where x_1 and x_2 is uniformly random and independent of x. The two shares of data are kept by P_1, P_2, respectively.

In the beginning phase, parties P_1, P_2 each enter input data input_1 and input_2, and locally compute two shared values $\mathsf{share}(\mathsf{input}_1)$ and $\mathsf{share}(\mathsf{input}_2)$, respectively. P_1, P_2 send one of the shared values to each other to perform other operations. In a secret sharing scenario, all data, including intermediate states, will be shared between the two participants. Intuitively, the two parties involved don't get any middle information. In order to support all possible operations, theoretically only addition and multiplication need to be supported, since these two operations are theoretically complete in terms of computational complexity. The following describes how to perform the computations for both operations with the premise of secret sharing.

- MpcAdd Party P_1 owns shared value (x_1, y_1), P_2 owns (x_2, y_2), where x_1, x_2 are the shared values of x, and y_1, y_2 are the shared values of y. With MpcAdd operation, P_1 and P_2 will obtain z_1 and z_2 respectively so that z_1 and z_2 are the shared values of $x + y$.

  It is easy to know that the only thing that P_1 and P_2 need to do is to compute $z_1 = x_1 + y_1$ and $z_2 = x_2 + y_2$ in local, respectively. No additional communication cost would take. The

correctness is assured by:

$$z_1 + z_2 = (x_1 + y_1) + (x_2 + y_2) = (x_1 + x_2) + (y_1 + y_2) = x + y$$

- MpcMul Party P_1 owns shared value (x_1, y_1), P_2 owns (x_2, y_2), where x_1, x_2 are the shared values of x, and y_1, y_2 are the shared values of y. With MpcAdd operation, P_1 and P_2 will obtain z_1 and z_2 respectively so that z_1 and z_2 are the shared values of $x \cdot y$.

  This operation is more complicated, we use the standard Beaver Triple method to implement it. It was proposed by cryptographer Donald Beaver in 1991[3]. With Rosetta's architecture, the helper node P_0 generates the triple (a, b, c), where a, b, c are uniformly random and subject to $c = a \cdot b$. P_0 computes shares of a, b, c locally into (a_1, b_1, c_1) and (a_2, b_2, c_2), and sends one share of a, b, c to P_1 and P_2 respectively. Then do the following.

P_1		P_2
Input: $(x_1, y_1, a_1, b_1, c_1)$		Input: $(x_2, y_2, a_2, b_2, c_2)$
Let $\delta_1 = x_1 - a_1$		Let $\delta_2 = x_2 - a_2$
$\epsilon_1 = y_1 - b_1$		$\epsilon_2 = y_2 - b_2$
	$\xrightarrow{\delta_1 = x_1 - a_1,\ \epsilon_1 = y_1 - b_1}$	
	$\xleftarrow{\delta_2 = x_2 - a_2,\ \epsilon_2 = y_2 - b_2}$	
set $\delta = \delta_1 + \delta_2, \epsilon = \epsilon_1 + \epsilon_2$		set $\delta = \delta_1 + \delta_2, \epsilon = \epsilon_1 + \epsilon_2$
$z_1 = c_1 + \delta \cdot b_1 + \epsilon \cdot a_1 + \delta \cdot \epsilon$		$z_2 = c_2 + \delta \cdot b_2 + \epsilon \cdot a_2$

It is easy to know that z_1, z_2 are the shares of $x \cdot y$ by checking:

$$\begin{aligned} z_1 + z_2 &= (c_1 + c_2) + \delta \cdot (b_1 + b_2) + \epsilon \cdot (a_1 + a_2) + \delta \cdot \epsilon \\ &= c + \delta \cdot b + \epsilon \cdot a + \delta \cdot \epsilon \\ &= (\delta + a) \cdot (\epsilon + b) \\ &= (x - a + a) \cdot (y - b + b) \\ &= x \cdot y \end{aligned}$$

In order to support different AI algorithms, other different operation will be supported, such as MpcMatMul, MpcReLU, MpcDiv, MpcSigmoid, etc. Each operation is implemented and optimized differently, and Rosetta will use SecureNN[15] first, and will later support different MPC algorithms.

3.3 Architecture of Rosetta

Rosetta benefits from the high scalability of TensorFlow. In TensorFlow, new operations can be customized and registered, as well as different implementations of this operation (Kernel). And can define and implement gradients (or derivatives) for each new operation, and bind them to that operation.

As mentioned earlier, TensorFlow separates the definition and execution of computation. When generating the computation graph, the gradient graph is automatically generated according to the gradient corresponding to each operation and the chain law of derivation. Therefore, in the TensorFlow framework, the developer only needs to write the model, instead of develop the back propagation process of model training. This is also what often refers to Automatic Differentiation.

Rosetta's overall architecture replicates the features of TensorFlow. The principle is to use C/C++ to implement basic operations and gradients, then convert the operations in the TensorFlow graph into privacy operations on demand, and finally perform the execution of privacy operations. The detailed architecture is shown in Figure 5

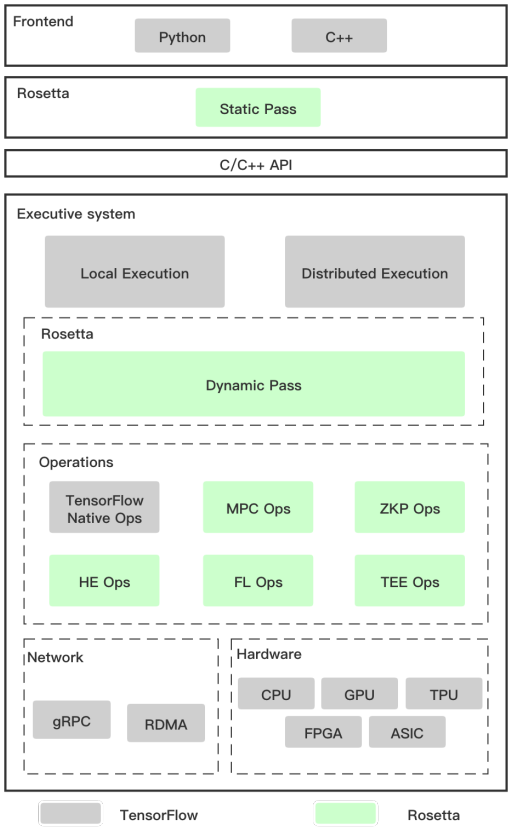

Figure 5: Architecture of Rosetta

As can be seen from the architecture, Rosetta reuses parts of functions and modules of TensorFlow, where the light green modules are Rosetta's own module, and the newly added parts are privacy operations, including MPC Ops, ZKP Ops, HE Ops, FL Ops and TEE Ops. These privacy operations represent the implementation of native operators in TensorFlow with corresponding privacy-preserving compution technologies, and will replace TensorFlow operation later.

Another additional part of Rosetta is Pass, which is a commonly used technology in compilers, mainly used for conversion and optimization. In Rosetta, it is mainly for the replacement of operations, as shown in Figure 6. For better scalability, Rosetta defines two Passes: Static Pass and Dynamic Pass, which are used for replacement and optimization at different stages. Details will be presented in the next section.

As mentioned above, the basic flow of Rosetta as a whole can be described as shown in Figure 7, which can be divided into the following main steps.

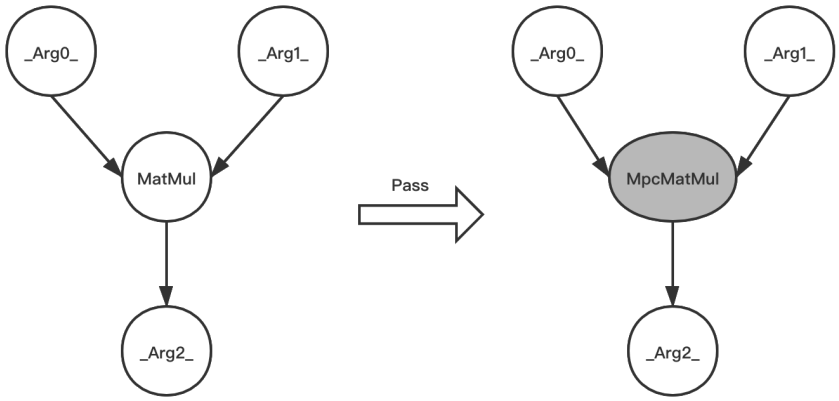

Figure 6:

1. Implements privacy operations and gradients in C/C++, and it is completely independent of the framework, and the existing implementations of various algorithms can be reused.
2. Register the privacy operations and gradients in TensorFlow and bind them. The privacy operations correspond to the native TensorFlow operations, and additionally provides privacy protection for the input and output.
3. The static directed graph in TensorFlow is replaced with privacy operations by Static Pass, which is called directed graph with privacy operations.
4. In the execution of the directed graph with privacy operation, it is performed by Dynamic Pass according to the TensorFlow execution process and outputs the final result.

As can be seen from the process, Rosetta has reused some of TensorFlow's modules, and therefore inherited TensorFlow's industrial-grade optimizations on these modules, including the following:

- **Directed Graph Compiler:** Rosetta fully reuses the directed graph compiled with TensorFlow, and thus it inherits the optimization of graph compilation therein.
- **Automatic Differentiation:** Instead of re-implementing the back propagation process, Rosetta uses automatic differentiation in TensorFlow to automatically replace operations in the gradient graph with privacy operation gradients. This significantly reduce the development time and difficulty for developers.
- **Graph Execution:** Rosetta fully reuses the process of graph execution in TensorFlow. It transforms the original operations to the corresponding privacy operations at the execution level. As a result, Rosetta is able to inherit various parallel optimizations in TensorFlow during the execution of the graph.

4 Core Modules

As can be seen from the architecture, the most important modules of Rosetta are the privacy operations and the Pass. This section focuses on the details of these two core modules. Since

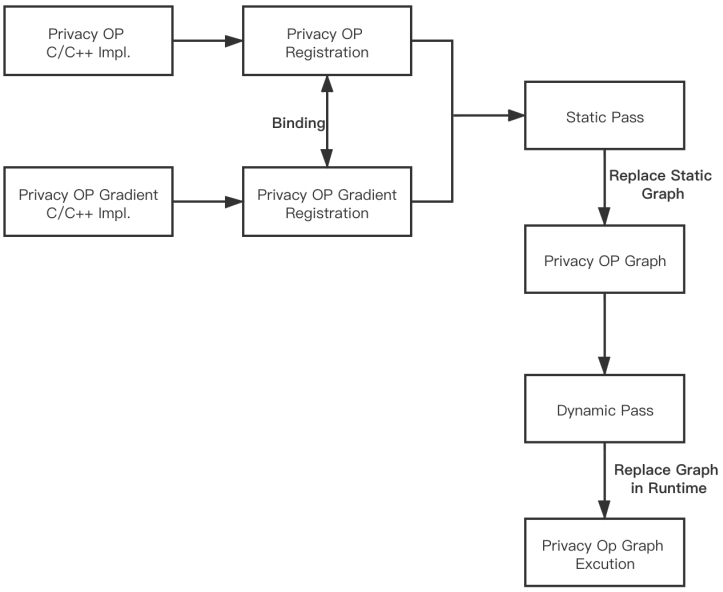

Figure 7: Workflow of Rosetta

Rosetta currently only supports MPC Ops, the MPC Ops are also used as an example in the following introduction.

4.1 MPC OP and MPC OP Gradient

The current version of Rosetta supports limited MPC operations, which are necessary to support logistic regression training. In other words, the basic function of the current version of Rosetta is to support training logical regression in a MPC manner. The Secure NN [15] algorithm is used in this release, and more MPC algorithms will be supported later.

Table 1 shows the operations and gradients that the current version of Rosetta will support, where '-' means there is no gradient exist.

There are more than 400 operators in TensorFlow, and other operations will be gradually supported in subsequent versions of Rosetta to enable training and inference of more complex AI models.

4.2 Pass

Pass is at the heart of connecting the TensorFlow framework to privacy-preserving computation in Rosetta. Pass was first conceptualized by the LLVM [10] compiler framework. The LLVM Pass framework is an important part of the LLVM system, and Pass performs the transformations and optimizations that make up the compiler, which builds the analytics used for these transformations. And most importantly, it is a structured technique that belongs to the compiler code.

In the Rosetta framework, the main purpose of Pass is to perform operation replacement. For better scalability, Rosetta defines two optimization passes: Static Pass and Dynamic Pass, which are used for replacement and optimization at different stages.

TensorFlow OP	MPC OP	MPC OP Gradient
Add	MpcAdd	MpcAddGrad
Sub	MpcSub	MpcSubGrad
Mul	MpcMul	MpcMulGrad
Div	MpcDiv	MpcDivGrad
TrueDiv	MpcTrueDiv	MpcTrueDivGrad
RealDiv	MpcRealDiv	MpcRealDivGrad
MatMul	MpcMatMul	MpcMatMulGrad
Sigmoid	MpcSigmoid	MpcSigmoidGrad
Log	MpcLog	MpcLogGrad
Log1p	MpcLog1p	MpcLog1pGrad
Pow	MpcPow	MpcPowGrad
Max	MpcMax	MpcMaxGrad
Mean	MpcMean	MpcMeanGrad
Relu	MpcRelu	MpcReluGrad
Equal	MpcEqual	-
Less	MpcLess	-
Greater	MpcGreater	-
LessEqual	MpcLessEqual	-
GreaterEqual	MpcGreaterEqual	-
SaveV2	MpcSaveV2	-
ApplyGradientDescentOp	MpcApplyGradientDescentOp	-

Table 1: Operation List

4.2.1 Static Pass

Recall the graph generation process of TensorFlow, after developers describes a model in Python, it will be converted into a directed graph by the compiler. If it is model training, it will also automatically convert into a gradient graph. Since TensorFlow's graph conversion is static, that is, the complete graph will be converted at once, Static Pass converts the operations in the static graph into privacy operations.

The Static Pass now consists of the following modules:

- CopyAndRepMpcOpPass. It is a specific Pass of Static Pass that performs data flow analysis of the TensorFlow static graph for all operations of the forward graph in the optimizer's minimize function. Based on the results of the data flow analysis, it decides whether to replace this operation with the MPC operations. If the data flow analysis confirms that it is an MPC operation, the original operation will be replaced. If the data flow analysis confirms that it is a local operation, it is sufficient to copy the operation directly. When the pass is performed, it will form another MPC-operation static graph with the same pattern as the previous static graph.

A replaceable MPC privacy optimizer (MpcOptimizer) is required for each optimizer in TensorFlow. There are a total of 11 optimizers in TensorFlow, and in the current version of Rosetta, only gradient descent optimizer is supported, and its corresponding privacy optimizer (MpcGradientDescentOptimizer) is implemented. Additional optimizers will be supported in subsequent releases.

4.2.2 Dynamic Pass

Dynamic Pass is the Pass that is performed when TensorFlow executes the graph, and in addition to the basic operation replacement, it is also introduced for Rosetta's subsequent scalability and flexibility. In the current version of Rosetta, Dynamic Pass contains the following modules.

- MpcSaveModelPass. It is a specific Pass of Dynamic Pass. It dynamically replace the SaveV2 operation with the MpcSaveV2 when saving model in TensorFlow.
- MpcOptApplyXPass. It is a specific Pass of Dynamic Pass. It updates weight variables according to the optimizer in training . Each optimizer in TensorFlow updates variables in a different way. Since the current version of Rosetta only supports the Gradient Descent Optimizer (GradientDescentOptimizer), this Pass currently only supports the corresponding training weights as follows:

$$\mathsf{Var} -= \eta \cdot \theta_{\mathsf{Var}}$$

Dynamic Pass provides Rosetta enough flexibility in terms of architecture. Subsequently, it is possible that changes due to the privacy-preserving computation algorithms upgrade or TensorFlow upgrade may affect the user's model, which may lead to problems in updating, installing and deploying the model again. In this case, thanks to Dynamic Pass, Static Pass does not need to be modified, which means the user does not have to re-update, install and deploy the model. Simply provide the appropriate version of the privacy operation and gradient implementation, and then replace the different operation with Dynamic Pass according to the version of TensorFlow.

Pass is borrowed from and migrated from the compiler's related technologies. Thus classical compiler optimization techniques such as constant folding, constant propagation, algebraic simplification, common sub-expressions elimination, etc. can also be applied to the Rosetta framework. At the same time, Dynamic Pass can further optimize the relevant computation for specific privacy-preserving computation algorithms.

5 Applications

Rosetta serves as the basic framework and tool to support a variety of products and a variety of application scenarios. This section will list and present some possible deployment scenarios. It is worth pointing out that to form a complete product and solution still requires on-demand design and development on top of Rosetta.

5.1 Collaborative Modeling

Collaborative modeling is the most classic pattern supported by Rosetta. In the actual scenario, the data is often distributed across different enterprises. There are various ways of distribution, such as horizontal distribution (different subjects, common attributes), vertical distribution (same subject, different attributes), etc. In order to be able to take advantage of the diversity of data and ensure the accuracy of model training, it is often necessary to integrate all data to train together. But to ensure data privacy, businesses are also reluctant to share data with each other.

Rosetta provides the basic development tools for collaborative modeling. Developers setup an AI model on TensorFlow, which is exactly the same development process as the existing way. During the training process, the model can be deployed on Rosetta and the MPC privacy-preserving computation manner can be initiated, and the model can be trained jointly between multiple parties without compromising their privacy data during the training process.

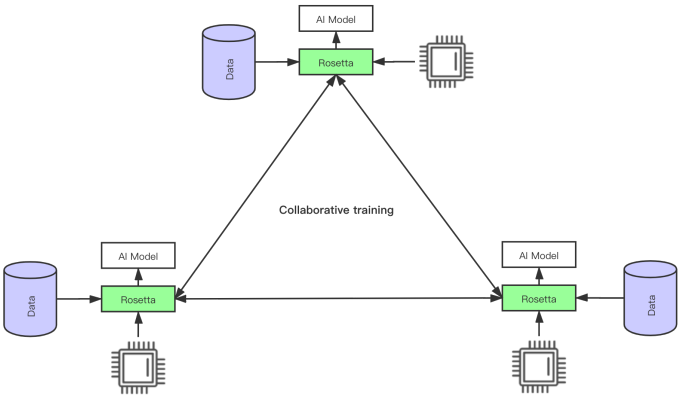

Figure 8: Collaborative modeling

5.2 Verifiable Open API

Open APIs have become a new Internet model designed to improve agility, execution and effectiveness. In various open API applications, enterprises provide various APIs for external queries. The API does not return all raw data to ensure data privacy, but generally returns the results of data processing, such as averages, variance, models trained from the data, etc. And in some scenarios, the caller of the API also wants to ensure the correctness and legality of the returned data through technical means. There is a certain "contradiction", that is, the user does not have the original data and cannot verify the correctness of the data returned by the API.

The problem of verifiable open APIs can be solved with the Rosetta framework. Developers can develop relevant API features on Rosetta that, when deployed, can initiate the ability to prove in zero knowledge. This capability allows companies to prove the correctness of the returned value for the calling party without giving explicit raw data.

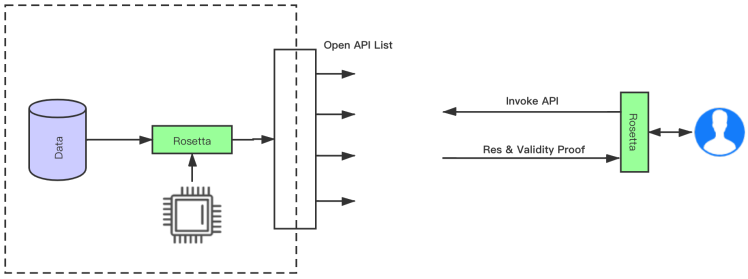

Figure 9: Verifiable Open API

5.3 Secure Data Outsourcing

With the rise of cloud computing, infrastructure costs for small and medium-sized enterprises have reduced significantly. Cloud computing centers are also gradually transforming into computing power centers. To reduce initial equipment costs, central enterprises often outsource data

to the cloud for computing. However, traditional cloud computing inevitably requires users to upload raw data to the cloud. This raises the "paradox" of data privacy and data outsourcing.

The Rosetta framework can solve the problem of secure data outsourcing. By using Rosetta to develop the corresponding processing logic, deployed on the client side and in the cloud, with the ability of homomorphic encryption, the cloud can only process the data in an encrypted state. The basic steps can be divided as follows.

- **Step 1:** The user encrypts the data to the cloud by invoking the ability of homomorphic encryption in Rosetta.
- **Step 2:** The cloud deploys the corresponding AI model through Rosetta and deploys the corresponding hardware acceleration capabilities. After receiving the encrypted data from the user, homomorphic operations are performed to obtain the resulting cipher and send it to the user. The computing process consumes a lot of computing power in the cloud, but no private data is revealed in the whole process.
- **Step 3:** The user decrypts the result via Rosetta and stores it in the database, preparing for the next application.

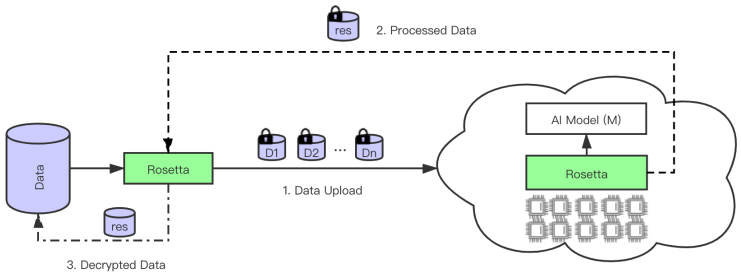

Figure 10: Secure Data Outsourcing

6 Conclusions

Rosetta is the first step towards realize privacy-preserving network of LatticeX Foundation, and it is still far from perfect. Therefore, with the power of the open source community, we hope to improve it through collective efforts, and welcome all kinds of opinions and criticisms. If you're an architect, we welcome ideas for various improvements to the Rosetta architecture. If you are an AI engineer or scientist, we welcome the use of Rosetta to develop complex models of all kinds. If you are an algorithm engineer, we welcome algorithms and mechanisms that integrate different kinds of privacy-preserving computaton technologies, including cryptography, federal learning, and trusted execution environments.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). pp. 265–283 (2016)

[2] ARM: TrustZone. https://developer.arm.com/ip-products/security-ip/trustzone

[3] Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference. pp. 420–432. Springer (1991)

[4] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., Zhang, Z.: Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274 (2015)

[5] Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)

[6] Costan, V., Devadas, S.: Intel sgx explained. IACR Cryptology ePrint Archive **2016**(086), 1–118 (2016)

[7] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing **18**(1), 186–208 (1989)

[8] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia. pp. 675–678 (2014)

[9] Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)

[10] Lattner, C., Adve, V.: Llvm: A compilation framework for lifelong program analysis & transformation. In: International Symposium on Code Generation and Optimization, 2004. CGO 2004. pp. 75–86. IEEE (2004)

[11] Lee, D., Kohlbrenner, D., Shinde, S., Song, D., Asanovic, K.: Keystone: A framework for architecting tees. CoRR **abs/1907.10119** (2019)

[12] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS-W (2017)

[13] Rivest, R.L., Adleman, L., Dertouzos, M.L., et al.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)

[14] Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: what it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/ISPA. vol. 1, pp. 57–64. IEEE (2015)

[15] Wagh, S., Gupta, D., Chandran, N.: Securenn: 3-party secure computation for neural network training. Proceedings on Privacy Enhancing Technologies **2019**(3), 26–49 (2019)

[16] Yao, A.C.: Protocols for secure computations. In: 23rd annual symposium on foundations of computer science (FOCS 82). pp. 160–164. IEEE (1982)