隐私 AI 开源框架—Rosetta 白皮书

Version 0.1

LatticeX Foundation

2020 年 7 月 29 日

目录

1 前言

数据是继土地、能源、人口、粮食之后的新一代生产要素，数据价值的体现一定是在数据使用过程中。数据与其他生产要素本质的区别在于，数据隐私是数据流动过程中产生价值的根本出发点。

目前互联网巨头商业模式与用户隐私保护间存在根本的矛盾。现有互联网公司商业模式的本质就是利用用户数据变现，在今天的互联网，人被异化成了数据的载体和工具。而新的全数字化时代，人必须回归个体的自然权利，将互联网平台“沦为”人的工具。

传统互联网巨头将受到其固有商业模式的限制，是很难进行左右互搏的。每一代的商业巨头都有自己的生命周期，传统互联网巨头将在隐私计算时代存在着不可跨越的危机。隐私计算一定是未来全数字时代的公共基础设施的重要支撑。

隐私 AI （Privacy-Preserving Artificial Intelligence）作为隐私计算的核心，奠基着人类在全数字化时代的生存方式。为不同主体和利益相关方达成一致，构造不同的交互范式，提供完备的、分布式的隐私保护解决方案。在计算复杂度与通讯复杂度之间的平衡，完美的支撑人类未来在数字经济体内的所有商业行为，这是 LatticeX 对隐私计算的理解与真正价值。

为支撑隐私 AI，亟需新的技术架构结合隐私计算技术与 AI 技术，以支撑上层产品与应用。这是 Rosetta 框架的基本出发点。

2 Rosetta 概述

人工智能（Artificial Intellegence，AI）技术和算法几乎应用在数据相关的各个场景。其中机器学习、深度学习通过收集大量的数据进行训练得出模型，再在不同的场景下利用模型对单个用户的数据进行处理。AI 技术在训练（training）和推断（inference）的过程中，涉及到各类数据的归集，而模型本身也是非常重要的数据。

由于 AI 技术的基本模式，导致目前遇到了极大的瓶颈，那就是数据隐私保护的问题。在数据量爆炸增长的今天，很难有一家机构收集到所有的数据。数据作为极为特殊的资产，企业间也不会真正彼此分享。而数据的多样性直接决定了模型的精度，进而决定了商业成本与风险。从 AI 算法的角度来看，数据越多越好；从隐私保护的角度来看，数据暴露的越少越好。这种内在的“矛盾”已经成为 AI 技术进一步使用最大的障碍。

隐私计算技术正是解决上述“矛盾”的根本性方法，隐私计算技术可以保证数据隐私的前提下进行 AI 模型的训练与推断。最大限度的利用数据的多样性，也最大限度的保证数据隐私。结合区块链作为分布式经济基础设施，最终形成完整的全数字时代的公共基础设施。

Rosetta 开源框架正是由此而生，连接 AI 技术与隐私计算技术，并有机的结合区块链智能合约。广为人知的罗塞塔石碑（Rosetta Stone）上记载着三种古文字：古埃及象形文字，埃及草书和古希腊文。Rosetta 因此寓意承载和结合着隐私计算，区块链和 AI 三种典型技

术，最终支撑起完备的隐私 AI 解决方案。

隐私计算（Privacy-Preserving Computation）技术大致可分为两类：基于算法（Trustless Computation）和基于硬件（Trusted Computation）。

基于算法的隐私计算技术以密码学为主，其中包含安全多方计算（secure Multi-Party Computation，MPC, [16]），零知识证明（Zero-Knowledge Proof，ZKP, [7]）和同态加密（Homomorphic Encryption，HE, [13]）。联邦学习（Federated Learning, [9]）则是从 AI 的视角和算法上来解决隐私保护的问题，也必不可免的使用各种密码学技术，因此也可归于基于算法的隐私计算技术。

基于硬件的隐私计算技术则是通过硬件打造一个可信执行环境（Trusted Execution Environment，TEE, [14]），在可信环境中，数据可以安全的使用和计算。较为典型的是 Intel 的 SGX [6] 和 ARM 的 TrustZone [2]，以及开源框架 KeyStone [11] 等等。

Rosetta 以支撑提供可部署、可运维、可运营的商用产品为最终目的。因此，采取的技术路径是结合常用的 AI 框架，以密码学算法为起点，并逐步扩展到其他隐私计算技术。架构如图 1 中红色虚线框所述。

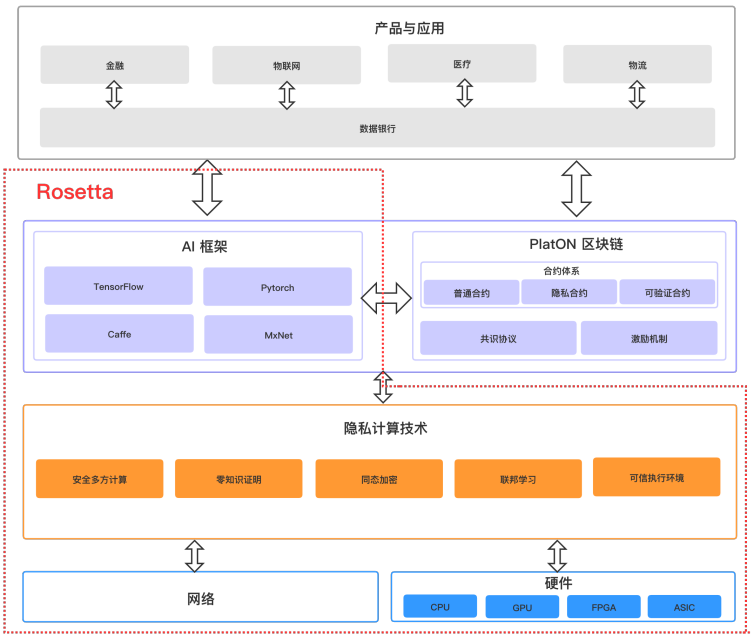

图 1: Rosetta 整体架构

- **网络与硬件** 隐私 AI 的场景一定是分布式的场景，因此必须有底层网络支撑。隐私计算技术大多对计算复杂度和通信复杂度要求较高，在解决网络通信的同时，也需能够通过硬件加速提升整体的运行速度，因此 Rosetta 以 CPU 为主，最终支持 GPU，FPGA 和 ASIC 等高性能的硬件加速设备。

- **隐私计算技术** Rosetta 将支持多种隐私计算技术，以密码学为主的安全多方计算，零知识证明，同态加密等；以 AI 算法为主的联邦学习以及与密码学算法的结合；同样也将支持 TEE 等相关技术。
- **AI 框架** 为支持完备的隐私 AI，吸引 AI 工程师可以低门槛或者无门槛地使用，Rosetta 将逐渐集成主流的 AI 框架，包括 TensorFlow [1]，Pytorch [12]，Caffe [8]，Mxnet [4] 等。目前阶段以 TensorFlow 为主。
- **产品与应用** Rosetta 最终支撑隐私 AI 的产品并为各行业的应用提供服务，并逐步服务于金融，物联网，医疗和物流等行业。

2.1 与区块链的关系

隐私计算网络与区块链作为数字化世界基础设施的两个重要组成部分，具有极其紧密的联系。隐私计算以实现数据资产化为最终目的，而数据资产化的基础为数据定价。显然，传统的以数据量大小为基准的定价方式已经不现实，也不准确。真正有价值的数据，并不在数据量的大小。LatticeX 认为数据的价值一定是体现为数据的效用（utility），真正被使用的数据才有价值。

基于隐私计算的数据交换、交易网络必须要有足够的数据流动性，也必然是分布式的，与区块链的分布式架构完全一致。而贡献隐私数据参与协同计算（不泄漏隐私）的初始动力，需要有激励机制支撑，这也是区块链机制能稳定运行的根本原因。最终，数据资产的清结算也一定是跨企业、跨行业的，只有作为分布式经济基础设施的区块链才能作为清结算的基本工具。

分布式、多中心的环境下，建立信任最有效的工具是区块链。通过区块链的共识算法对数据的状态、数据的操作流程达成最终一致。通过智能合约承载各类隐私计算应用，最终形成完备的隐私数据交易市场。

因此，只有区块链网络和隐私计算网络结合才能真正支撑起数据资产化的全流程。

2.2 Rosetta 的优势

Rosetta 开源框架是面向 AI 开发者、AI 研究人员的工具。同样也需要了解、熟悉隐私计算技术的开发者，研究人员来支撑和完善。因此 Rosetta 针对这两类潜在用户，在架构上既充分解耦，发挥各自领域的优势；又能有机的结合形成一个完备的整体，以达到快速迭代和优化目的。因此，Rosetta 具备以下三个重要的特点：

- **易用性** 密码学与 AI 学科之间有着较大的差距。特别是密码学，涉及到较为复杂、抽象的数学工具，学习曲线较高。因此让 AI 工程师完全熟悉和掌握复杂的密码学算法，会导致使用门槛极高。另一方面，AI 算法复杂多样，也无法通过密码算法来逐个定制化。这样既不利于开发者使用，也不利于后续的扩展。

基于此，Rosetta 集成常用的 AI 框架。当前阶段以市场占用量最大的 TensorFlow 为主。利用隐私计算算法将框架中的各类 AI 操作/算子（Operation）转化为带隐私保护的算子，即功能不变，附带保护数据隐私。这样，AI 工程师/研究人员即可按照传统的 TensorFlow 的使用方式，利用基础隐私算子拼装出所需模型。

为了进一步降低开发者的使用门槛，Rosetta 完全保留了 TensorFlow 的接口 API。AI 开发者可以在明文状态的利用 TensorFlow 来调整模型，一旦需要引入多方数据进行处理，只要简单的引入下面的包即可自动转化为隐私状态下的计算。

```
import latticex.rosetta
```

- **高效性** 隐私计算算法对于性能有极高的要求。以密码学算法为例，由于其使用的独特的数学模块，使得密码学研究人员更倾向于使用高效的编程语言来实现，比如 C/C++。虽然 TensorFlow 底层的实现也是采用 C/C++，但是两者的目的有本质的不同。密码学需要有更快速的针对代数运算的工具库，而 TensorFlow 底层的实现更多的是针对数值运算做的优化。因此直接采用 TensorFlow 来实现密码算法会有较多的限制，也会带来一定的性能瓶颈。另外，由于大多数密码算法都是基于 C/C++ 实现，因此将其转为 TensorFlow 实现，必然会带来算法重构的巨大工作量。

  另一方面，作为工业级产品，TensorFlow 在很多层面做了工业级的优化。比如子图的拆分，图的并行执行等等，这些都是可以完全复用在隐私算子的执行过程中。

  基于此，Rosetta 依然保持底层 C/C++ 实现的密码算法的高性能，同时有机的结合 TensorFlow 框架中对于并行计算的工业级优化，在保持易用性的前提下，依然保证算法的高效性。

- **灵活性** 对同一个隐私算子，存在多种密码学算法的实现。每个算法有独有的优势，适用于不同的场景。即使是同一个场景也有不同的密码学算法来匹配不同的部署环境。

  基于此，Rosetta 可灵活的支持各类隐私计算算法，包括密码学的安全多方计算，零知识证明和同态加密及其中的各类算法，同时也将支持联邦学和可信执行环境。以密码算法为例，在安全多方计算的实现中，将首先集成各类 3 方协议，并逐步扩充到 n 方的协议。密码算法的实现可以完全独立于框架，这样可为密码学开发者提供足够的灵活度。同时，Rosetta 提供丰富的接口便于算法与框架的集成。

2.3 Rosetta 路线图

按照技术的成熟度与需求，Rosetta 制定了如下的路线图，如图2 所示。

- **开源发布 (2020Q2)**

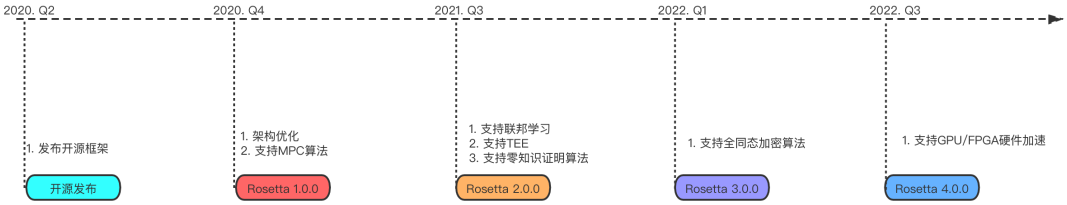

图 2: Rosetta 里程碑

Rosetta 的整体架构将在 2020 年四月底五月初正式开源。该版本完成与 TensorFlow 结合的基本框架设计，集成安全多方计算协议，可以实现简单的机器学习算法的隐私训练和推断。

- **Rosetta v1.0.0 (2020Q4)**

  Rosetta 1.0.0 版本将在 2020 年第四季度正式发布。该版本完成架构和算法的优化，并支持 TensorFlow 中所有算子转化为隐私算子。

- **Rosetta v2.0.0 (2021Q3)**

  Rosetta 2.0.0 版本将在 2021 年的第三季度发布。其中会实现和集成其他的隐私计算算法，包括联邦学习和可信执行环境。同时，也会支持密码学的零知识证明算法。

- **Rosetta v3.0.0 (2022Q1)**

  Rosetta 3.0.0 版本将在 2022 年第一季度发布，主要的功能是集成全同态加密算法。针对 AI 算法中对于数值计算的要求，该版本中将会采用适合数值计算的 CKKS [5] 方案。

- **Rosetta v4.0.0 (2022Q3)**

  Rosetta 4.0.0 版本是相对完善的软硬件结合版本，将支持和集成各类硬件加速方案，以降低整体的计算复杂度。

3 Rosetta 总体架构

在该节中将主要介绍 Rosetta 的架构和原理的细节，为了更好地理解架构设计的初衷，会简要介绍 TensorFlow 的架构，以及隐私计算算法的原理。隐私计算算法以安全多方计算（MPC）算法为例。

3.1 TensorFlow 架构概述

TensorFlow 采用数据流图（Data Flow Graphs）进行数值计算，如图 3 所示。在图中，节点（Nodes）表示数学操作，比如矩阵加法，矩阵乘法等。图中的线（Edges）表示节点间相互联系的多维数据数组，即张量（Tensor）。节点和线构成一个有向图（Directed Graph），线的方向表示数据的输入输出，张量从图中流过（Flow）表示对数据执行操作，这也是 TensorFlow 名字的来源。

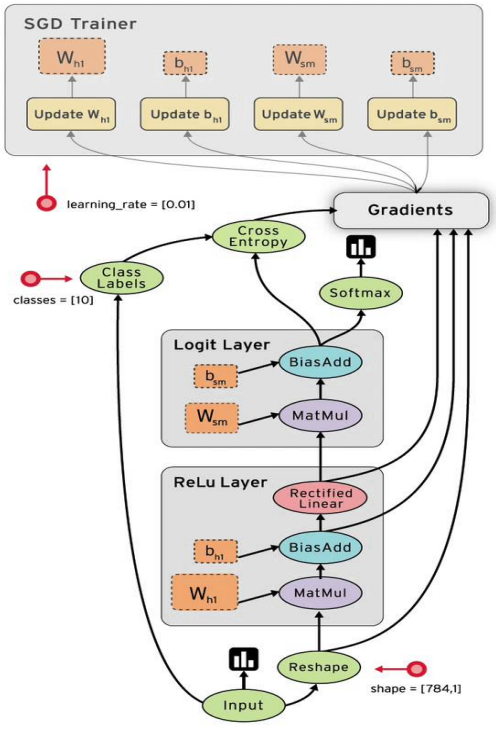

图 3: 数据流图（摘自 http://www.tensorfly.cn/）

TensorFlow 将计算的定义与其执行分开。前端允许开发者采用门槛较低的编程语言（Python）定义需要进行的计算，比如 AI 模型等。TensorFlow 会自动将其转换为有向图。如下的代码块用 Python 描述了整个计算过程，图 4 所示为自动转化的有向图。

```
import tensorflow as tf

b = tf.Variable(tf.zeros([128]), name="b")                    # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,1],-1,1), name="w")    # 784x100 matrix w/rnd vals
x = tf.Variable(tf.random_uniform([128, 784]), name="x")      # Placeholder for input
relu = tf.nn.relu(tf.matmul(x, W) + b)                        # Relu(Wx+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()
```

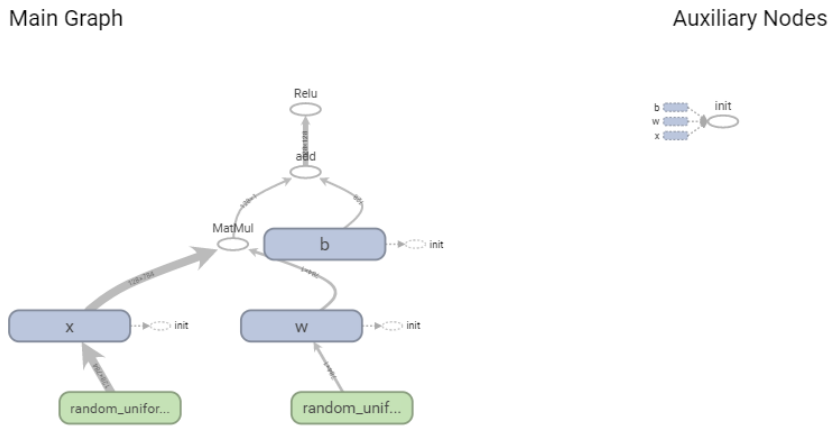

图 4: TensorBoard 生成的有向图

有向图的节点表示一个操作（Operation），该操作在 TensorFlow 的内核（Kernel）中用 C/C++ 实现。在给定输入数据后，后端按照有向图的结构执行整个计算。在图的生成和计算的执行过程中，TensoFlow 做了大量的工业级的优化，包括图/子图的结构，并行执行等优化。

Rosetta 架构将会尽可能的利用 TensorFlow 现有的工业级优化。

3.2 MPC 概述

安全多方计算（MPC）[16] 最初由姚期智先生提出，旨在多个参与方之间不泄漏各自隐私输入的前提下完成计算。实现安全多方计算的协议有很多种，这里主要介绍基于秘密分享（Secret Sharing）的算法。

为简单起见，本文考虑两个参与方的情况（P_1, P_2），但是有一个帮助节点（Helper Node，P_0）用于随机数生成。本文只考虑半诚实（Semi-Honest）模型安全模型，更为复杂的恶意（Malicious）模型可参考相关论文。另外，本文只考虑多数诚实者（Honest-Majority）模型，也就是说上述三个节点不会两两串谋。本文考虑 uint64_t 的数据类型。

给定一个数 x，定义 $\mathsf{share}(x) = (x - r, r)$，其中 x 是随机数。可以得出，如果 $\mathsf{share}(x) = (x_1, x_2)$，那么有 $x = x_1 + x_2$。其中单一的 x_1 或者 x_2 都是完全随机，并且与 x 独立。数据分享的两部分将会由两个参与方各自保存。

在开始阶段，参与方 P_1, P_2 各自将输入数据 input_1 和 input_2 本地计算得到两个分享值 $\mathsf{share}(\mathsf{input}_1)$ 和 $\mathsf{share}(\mathsf{input}_2)$，并将其中的一个分享值发送给对方，以此为起点来执行各种不同的操作。在秘密分享的方案中，所有的数据，包括中间状态都将会分享在两个参与方之间。直观的看，参与的两方不会得到任何的中间信息。为了支持所有可能的操作，理论上来讲只需要支持加法和乘法，因为这两个操作在计算复杂性理论上来说是完备的。下面介绍如何在秘密分享的前提下进行这两个操作的计算。

MpcAdd P_1 拥有分享值 (x_1, y_1)，P_2 拥有分享值 (x_2, y_2)。其中 x_1, x_2 是 x 的分享值，y_1, y_2 是 y 的分享值。该操作最后要使得 P_1 和 P_2 分别计算得到 z_1 和 z_2，使得 z_1, z_2 是 $x + y$ 的分享值。这样才可以进行后续的其他操作。

很容易得知，P_1, P_2 分别本地计算 $z_1 = x_1 + y_1$，$z_2 = x_2 + y_2$ 即可。该过程既不需要通信，计算过程也非常简单。而且正确性可由下可知：

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

MpcMul P_1 拥有分享值 (x_1, y_1)，P_2 拥有分享值 (x_2, y_2)。其中 x_1, x_2 是 x 的分享值，y_1, y_2 是 y 的分享值。该操作最后要使得 P_1 和 P_2 分别计算得到 z_1 和 z_2，使得 z_1, z_2 是 $x \cdot y$ 的分享值。这样才可以进行后续的其他操作。

该操作相对较为复杂，我们采用最为经典的 Beaver Triple 方法。该方法是由密码学家 Donald Beaver 在 1991 年提出 [3]。在本文的框架下，可先由帮助节点 P_0 生成一个 (a, b, c) 三元组（Triple），其中 a, b, c 为随机数，并且满足 $c = a \cdot b$。P_0 将 a, b, c 本地分享成 (a_1, b_1, c_1) 和 (a_2, b_2, c_2)，并且把两组分享值分别发给 P_1 和 P_2。然后进行如下协议：

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

容易得知，z_1, z_2 是 $x \cdot y$ 的分享值。这是因为：

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

为了支持不同的 AI 算法，需支持其他不同的算子，比如 MpcMatMul，MpcReLU，MpcDiv，MpcSigmoid 等等。每一个算子都有不同的实现和优化，Rosetta 当前版本将会采用 SecureNN[15] 中的算法，后续也将支持不同的 MPC 算法。

3.3 Rosetta 架构概述

Rosetta 得益于 TensorFlow 的高可扩展性。在 TensorFlow 中，可以自定义和注册新的操作（Operation），以及该操作的不同实现方式（Kernel）。并且可为每一个新的操作定义和实现梯度（Gradient，或者说导数），并将其与该操作进行绑定。

如前所述，TensorFlow 是将计算的定义和执行分开的。在生成计算的有向图时，根据每个操作对应的梯度，以及求导的链式法则，会自动生成梯度图。因此在 TensorFlow 框架下，开发者只需要编写模型即可，而不需要再逐步去编写训练过程中的反向传播过程。这也是工业界常说的自动求导过程。

Rosetta 的整体架构复用了 TensorFlow 的特性，基本的模式为利用 C/C++ 实现基本的算子以及梯度，然后将 TensorFlow 的图中的算子按需转换成为隐私算子，最后进行隐私算子的执行。详细架构图如图 5 所示。

从架构图可以得知，Rosetta 复用了 TensorFlow 部分功能和模块，浅绿色的模块为 Rosetta 自有的模块。其中新增的部分是隐私算子，包括 MPC 算子，ZK 算子，HE 算子，FL 算子和 TEE 算子。这些隐私算子表示通过不同的隐私计算技术来实现 TensorFlow 中原生的算子，并且将会在后续替换 TensorFlow 的算子。

Rosetta 另外一部分新增的内容是优化遍（Pass）。优化遍（Pass）是编译器中常用的技术，主要用作转化和优化。在 Rosetta 框架中，其主要目的是进行算子的替换，如图 6 所示。为了具有更好的可扩展性，Rosetta 定义了两种优化遍：静态优化遍（Static Pass）和动态优化遍（Dynamic Pass），分别用于不同的阶段进行替换和优化。其细节将在下一节中介绍。

如上所述，可以将 Rosetta 整体的基本流程描述成如图 7 所示，按照先后顺序主要可以分成以下几步：

1. 采用 C/C++ 实现隐私算子及其梯度函数。该实现过程可以完全和框架独立，也因此可以复用现有的各种算法实现。

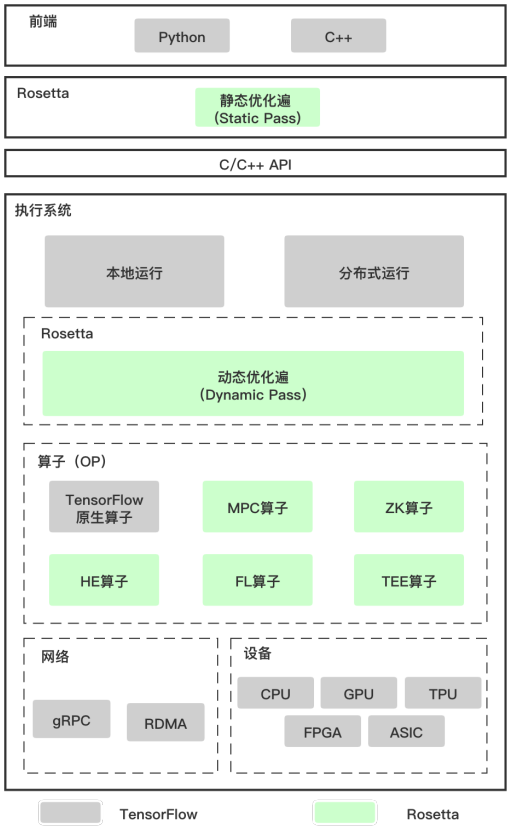

图 5: Rosetta 详细架构图

2. 在 TensorFlow 中注册隐私算子和梯度函数，并将两者绑定。隐私算子的功能与原生的 TensorFlow 的算子对应，并且额外提供对每个算子输入输出的隐私保护能力。

3. 通过静态优化遍（Static Pass）将 TensorFlow 中的静态有向图按照一定规则替换为由隐私算子组成的有向图，将其称为隐私算子有向图。

4. 在隐私算子有向图的执行过程中，按照 TensorFlow 执行过程以及动态优化遍（Dynamic Pass）执行隐私算子，最终得到输出结果。

从流程可以看出，Rosetta 复用了部分 TensorFlow 的模块，因此也继承了 TensorFlow 在这些模块上的工业级优化，包含以下几个方面：

- **复用有向图编译器**。Rosetta 完全复用 TensorFlow 编译的有向图，因此也就继承了其中对于图编译的优化。

- **复用自动求导**。Rosetta 不再重新实现反向传播过程，而是利用 TensorFlow 中自动求

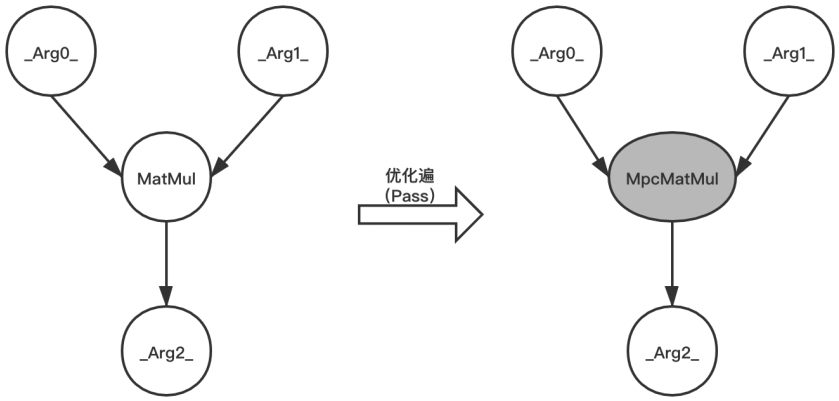

图 6: 算子替换

导的功能，将梯度图中的算子利用隐私算子梯度进行自动替换。因此能够大量减少开发者的开发时间和难度。

- **复用图执行过程**。Rosetta 完全复用 TensorFlow 执行图的过程，只是在执行层面上将原来的操作转换为对应的隐私算子的操作。因此，Rosetta 能够继承 TensorFlow 中在图的执行过程中的各种并行的优化。

4 Rosetta 核心模块

从架构图可以看出，Rosetta 最为重要的模块是隐私算子和优化遍（Pass）。该节中主要介绍这两个核心模块的细节。由于 Rosetta 当前版本只支持部分安全多方计算算子，即 MPC 算子，因此后面的介绍也以 MPC 算子为例。

4.1 MPC OP 和 MPC OP Gradient

Rosetta 当前版本支持部分 MPC 算子，算子是支撑逻辑回归（Logistic Regression）训练所必须的。换言之，Rosetta 当前版本的基本功能为支持逻辑回归 MPC 的方式下进行训练。该版本中采用 Secure NN [15] 的算法，并将在后续支持更多的 MPC 算法。

如下表 1 是 Rosetta 当前版本中将支持的算法以及其梯度，其中 - 表示不存在梯度。

TensorFlow 中一共有 400 多个算子，在 Rosetta 的后续版本中将逐渐支持其他算子，以支持更复杂的 AI 模型的训练和预测。

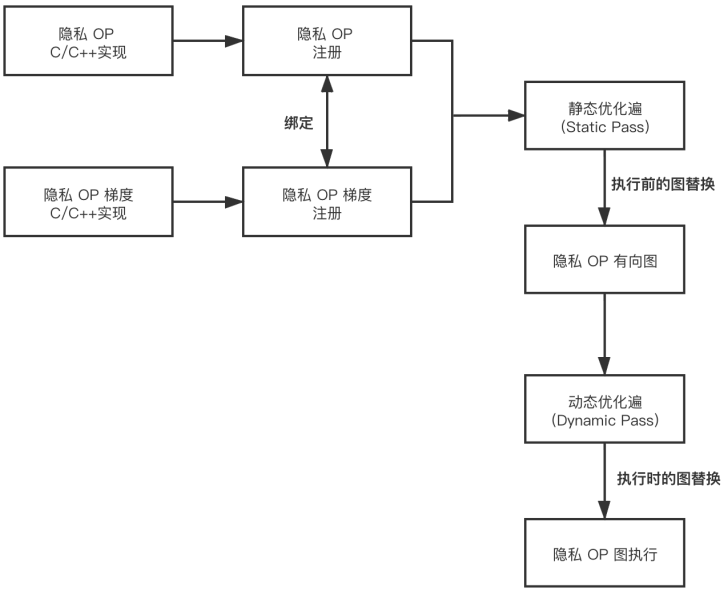

图 7: 整体流程图

4.2 优化遍 Pass

优化遍（Pass）是 Rosetta 中连接 TensorFlow 框架与隐私计算的核心。Pass 最早是由 LLVM [10] 编译器框架提出的概念。LLVM Pass 框架是 LLVM 系统的重要组成部分，Pass 执行构成编译器的转换和优化，它构建这些转换所使用的分析结果。并且最重要的是，它是属于编译器代码的结构化技术。

在 Rosetta 框架中，Pass 的主要目的是进行算子的替换。为了具有更好的可扩展性，Rosetta 定义了两种优化遍：静态优化遍（Static Pass）和动态优化遍（Dynamic Pass），分别用于不同的阶段进行替换和优化。

4.2.1 静态优化遍（Static Pass）

回顾一下 TensorFlow 的图生成过程，在开发者使用 Python 描述模型之后，会通过编译器将其转化为有向图。如果是模型训练，还会自动求导转化成梯度图。因为 TensorFlow 的图转换是静态的，也就是会一次转换完整的图，因此静态优化遍（Static Pass）就是将该静态图中的算子，转换为隐私算子。

静态优化遍主要由下面的模块完成：

CopyAndRepMpcOpPass 为 Static Pass 中的一种特定的 Pass，该 Pass 的作用是对 TensorFlow 静态图在优化器的 minimize 函数中对前向图所有算子进行逐一数据流分析。

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	-

表 1: 算子列表

根据数据流分析结果判断是否要把该算子替换为 MPC 隐私算子。如果数据流分析确认是 MPC 隐私操作，则需把原算子替换为 MPC 隐私算子。如果数据流分析确认是本地操作，则直接深度拷贝该算子即可。该 Pass 执行完毕后就会形成跟之前静态图同样形态的另外一个 MPC 隐私算子静态图。

针对 TensorFlow 中的优化器（Optimizer）都需要一个可替换的 MPC 隐私优化器（MpcOptimizer）。在 TensorFlow 中一共有 11 个优化器，在 Rosetta 当前版本中，只支持较为常用的梯度下降优化器，并实现其对应的隐私优化器（MpcGradientDescentOptimizer）。其他的优化器将在后续版本中支持。

4.2.2 动态优化遍（Dynamic Pass）

Dynamic Pass 是指在 TensorFlow 执行图的时候所执行的 Pass。除了基本的算子替换之外，其引入也是为了 Rosetta 后续的可扩展性和灵活性而设计。在 Rosetta 当前版本中，Dynamic Pass 包含如下模块:

MpcSaveModelPass 为 Dynamic Pass 中的一种特定的 Pass。该 Pass 的主要作用是在

TensorFlow 运行模型保存的过程中动态替换 SaveV2 算子为 MpcSaveV2 隐私算子。

MpcOptApplyXPass 为 Dynamic Pass 中的一种特定的 Pass。该 Pass 的主要作用是更新优化器训练权重变量。TensorFlow 中每种优化器更新变量的方式都不一样，由于 Rosetta 当前版本只支持梯度下降优化器（GradientDescentOptimizer），因此该 Pass 目前只支持对应的训练权重的更新方式，如下所示：

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

Dynamic Pass 在架构上为 Rosetta 提供了足够大的灵活性。在后续，有可能因为隐私算法升级或者 TensorFlow 升级带来的算子变更对用户的模型带来影响，由此会带来重新更新、安装和部署模型的问题。在这种情况下，由于 Dynamic Pass 的存在，Static Pass 无需修改，也就意味的用户无需重新更新、安装和部署模型。只要提供相应版本的隐私算子及梯度实现即可，然后在 Dynamic Pass 中根据 TensorFlow 的版本替换不同的算子即可。

Pass 是从编译器的相关技术中借鉴和迁移过来的。因此传统的类编译器的优化技术，比如常量折叠、常量传播、代数化简、公共子表式消除等，也可应用到 Rosetta 框架之中。同时，针对特定的隐私计算算法，Dynamic Pass 也可以进一步优化相关的计算。

5 Rosetta 部署场景

Rosetta 作为基本的框架和工具，可支持各类产品以及多种应用场景。本节将列举和介绍一些可能的部署场景。值得指出的是，要形成完备的产品和解决方案仍然需要在 Rosetta 之上按需进行设计和开发。

5.1 联合建模

联合建模是 Rosetta 支持的最为经典的模式。在实际场景中，数据往往分布在不同的企业中。分布的方式可以多种多样，比如横向分布（不同主体，共同的属性），纵向分布（相同主体，不同属性）等等。为了能够利用数据的多样性，保证模型训练的精度，往往需要整合所有数据一起训练。但是为保证数据隐私，企业彼此也不愿意共享数据。

Rosetta 为联合建模提供了基本的开发工具。开发者可以在 TensorFlow 上开发出 AI 模型，该开发过程与现有的模式完全一样。在训练过程中，可以将该模型部署在 Rosetta 上，启动 MPC 隐私计算模式，则可以在多方之间进行联合的模型训练，训练过程中不会泄漏各自的隐私数据。

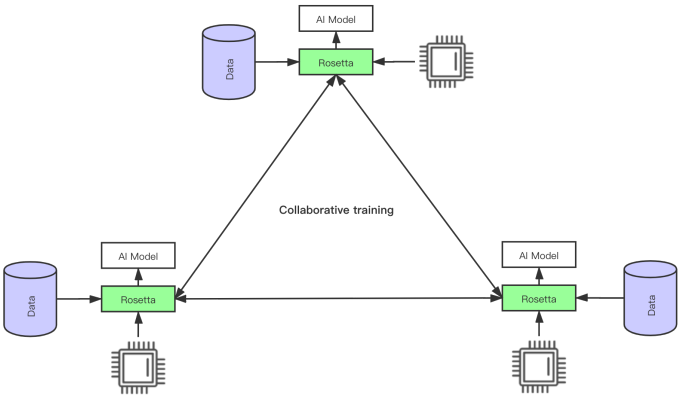

图 8: 联合建模

5.2 可验证开放 API

开放 API 已经成为一种新的互联网模式，旨在提高敏捷性，执行效率和有效性。在各类开放 API 应用中，企业提供各种 API 供外部查询。为保证企业的数据隐私，API 并不会返回所有原始数据，而一般是返回数据处理后的结果，比如平均值，方差，数据训练的模型等等。而在某些场景中，API 的调用方也希望通过技术手段确保返回数据的正确性与合法性。这样就存在一定的“矛盾”，即用户没有原始数据也无法验证 API 返回数据的正确性。

通过 Rosetta 框架可以解决可验证开放 API 的问题。开发者可以在 Rosetta 上开发相关的 API 功能，部署后，可以启动零知识证明的能力。该能力允许企业在不给出明文原始数据的前提下，为调用方证明返回值的正确性。

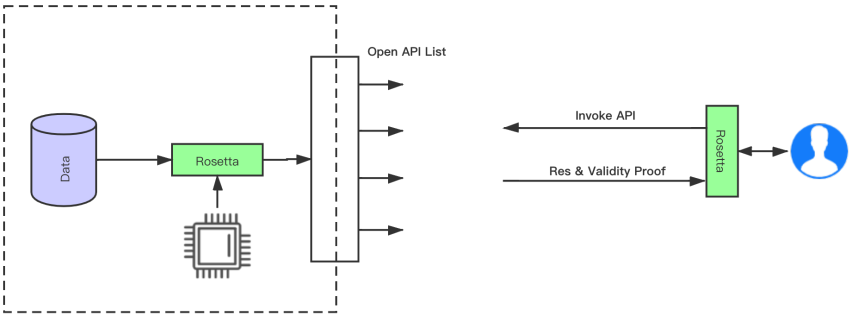

图 9: 可验证开放 API

5.3 安全数据外包

随着云计算的兴起，中小企业的基础设施成本大量降低。云计算中心也逐渐转变为算力中心。为减少初期设备成本，中心企业往往将数据外包到云上进行计算。然而传统的云计算避不可免的需要用户将原始数据上传到云上。因此引发了数据隐私与数据外包的“矛盾”。

Rosetta 框架可以解决安全数据外包的问题。通过利用 Rosetta 开发相应的处理逻辑，部署在客户侧和云端，在开启同态加密的能力后，云计算中心则可以在密文的状态下对数据进行处理。基本的步骤可以分为：

第一步 用户通过调用 Rosetta 中的同态加密的能力将数据加密传输到云端。

第二步 云端通过 Rosetta 部署相应的 AI 模型，并且部署相应的硬件加速能力。在接收到用户的加密数据后，进行同态运算，得到结果的密文并将该密文发送给用户。计算过程中，云端会消耗大量的计算能力，但是全过程对数据不可见。

第三步 用户通过 Rosetta 将结果解密，存储在数据库中，准备下一步的应用。

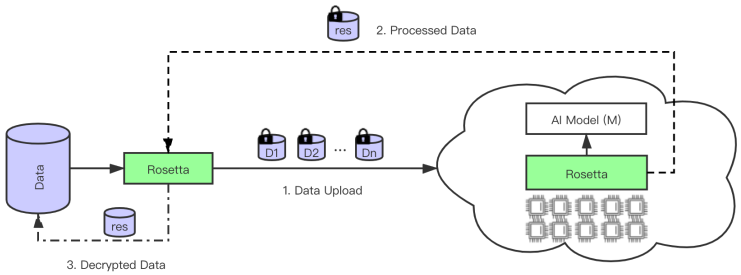

图 10: 安全数据外包

6 写在后面

Rosetta 是为实现 Latticex Foundation 隐私计算走出的第一步，依然远非完美。故此借助开源社区的力量，希望群策群力加以完善，也欢迎各种意见和批评。如果你是架构师，我们欢迎对 Rosetta 架构的各类改进意见。如果你是 AI 工程师，我们欢迎使用 Rosetta 开发各类复杂的模型。如果你是算法工程师，我们欢迎集成各类不同的隐私计算的算法和机制，包括密码学，联邦学习，可信执行环境。

参考文献

[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)