



中山大學 网络空间安全学院

SUN YAT-SEN UNIVERSITY SCHOOL OF CYBER SCIENCE AND TECHNOLOGY

# 数据安全与隐私保护

## CSE 309 第三次作业

作业内容： 隐私计算的开源框架调研

完成时间： 2023 年 12 月 17 日

学院： 网络空间安全学院

专业： 网络空间安全

# 目录

<b>1. 实验原理</b>	<b>1</b>
1.1. SPDZ 协议	1
1.1.1. Beaver 协议	1
1.1.2. 基于 Beaver 的 SPDZ 协议	2
1.2. PySyft 简介	3
1.3. 安全多方计算框架	5
1.3.1. 建立一个 MPCTensor	5
1.3.2. 应用差分隐私	5
<b>2. 实验内容</b>	<b>7</b>
2.1. 实验要求	7
2.2. 实验设计	8
2.3. 实验步骤	8
<b>3. 实验分析</b>	<b>9</b>
3.1. 整体示意	9
3.2. PySyft 组件	9
3.3. PySyft 概念	10
3.4. 核心代码解读	11
3.4.1. Syft 函数	11

3.4.2. 设计 Policy .....	12
<b>4. 实验记录 .....</b>	<b>17</b>
4.1. 数据拥有者上传私有数据集 .....	17
4.1.1. 启动 Syft 领域服务器 .....	17
4.1.2. 添加数据主体 .....	17
4.1.3. 准备数据集 .....	18
4.1.4. 创建 Syft 数据集 .....	19
4.1.5. 上传 Syft 数据集到领域服务器 .....	20
4.2. 数据分析者提交代码 .....	21
4.2.1. 查看领域服务器的数据集 .....	21
4.2.2. 创建 Syft 函数 .....	23
4.2.3. 向领域服务器提交代码 .....	24
4.2.4. 运行 Syft 函数 .....	25
4.3. 数据拥有者批准代码的执行 .....	25
4.3.1. 查看信息 .....	25
4.3.2. 运行代码 .....	27
4.3.3. 处理请求 .....	27
4.4. 数据分析者查看代码执行的结果 .....	28
<b>附录 .....</b>	<b>30</b>
附录 A: 参考文献 .....	30

# 1. 实验原理

## 1.1. SPDZ 协议

SPDZ (Secure Protocol for Distributed Zero-Knowledge) 协议是一种高度通用的多方计算协议, 适用于各种计算任务, 包括机器学习模型的训练和评估等。

SPDZ 协议使用零知识证明来确保各方在计算中不需要知道对方的输入, 只需知道计算的结果。SPDZ 协议的实现涉及密码学技术, 如同态加密和多方计算协议。这些技术允许在不公开原始输入的情况下执行计算。

对于算术电路来说, 只需要“加法门”与“乘法门”就是完备的, 其它的电路门都可以通过“加法门”与“乘法门”表达出来。因此针对算数电路设计安全多方计算协议, 需要解决“加法门”与“乘法门”的计算, 其中加法的实现比较容易实现, 相对乘法实现就没有那么简单了。

下面将介绍基于半诚实敌手模型下 Beaver 协议的 SPDZ 协议的工作原理, 来说明 SPDZ 协议。SPDZ 协议最早由 Damgrd I 等在文章《Multiparty Computation from Somewhat Homomorphic Encryption》提出, 它把协议分为了预处理阶段与在线阶段两个阶段: 预处理阶段根据 Beaver 协议生成乘法三元组为在线阶段做准备, 在线阶段进行秘密分享消耗预处理阶段生成的三元组。

### 1.1.1. Beaver 协议

#### 加法门

加法的秘密分享方案很简单, 将隐私数据拆分成  $x = x_1 + x_2$  和  $y = y_1 + y_2$ ,  $x_1$  和  $y_1$  发送给  $P_1$ ,  $x_2$  和  $y_2$  发送给  $P_2$ , 这样就完成了输入值的分享。

对于加法门来说只要  $P_1$  本地计算  $x_1 + y_1$ ,  $P_2$  本地计算  $x_2 + y_2$ , 就完成了加法门输出值  $x + y$  的秘密分享, 而不会泄露隐私数据  $x$  和  $y$  的信息, 且整个过程并不需要交互。

#### 乘法门

乘法门的秘密分享相对加法就没有那么简单了, 因为  $xy = (x_1 + x_2)(y_1 + y_2) = x_1y_1 + x_1y_2 + x_2y_1 + x_2y_2$ , 这个时候, 无论怎么把各个计算任务分配给两个参与方  $P_1$  和  $P_2$ , 都需要同时交换  $y_1$  和  $y_2$  的值, 从而泄露  $y$  的信息。

Beaver 提出了一种基于秘密分享的针对算术电路实现的安全多方计算协议。每个“乘法门”的计算都需要消耗一个 Beaver 三元组  $(a, b, c)$ , Beaver 三元组是随机数构成的并不涉及到计算的真实输入, 其中  $c = ab$ 。在预处理阶段需要提前计算出大量的 Beaver 三元组以备。

两个参与方  $P_1$  和  $P_2$  分别拥有的三元组  $(a_1, b_1, c_1)$  和  $(a_2, b_2, c_2)$ , 其中  $a_1, b_1$  表示  $P_1$  拥有  $a$  和  $b$  的秘密分享值,  $a_2, b_2$  表示  $P_2$  拥有  $a$  和  $b$  的秘密分享值, 参与方的三元组满足  $a = a_1 + a_2, b = b_1 + b_2, c = c_1 + c_2$ , 同时  $c = ab$ , 即

$$ab = (a_1 + a_2)(b_1 + b_2) = c_1 + c_2$$

每个参与方参与方分别只有  $a, b, c$  部分份额, 并都不知道  $(a, b, c)$ , 可以使用 Paillier 同态加密方案构造出符合的三元组过程如错误!未找到引用源。所示。

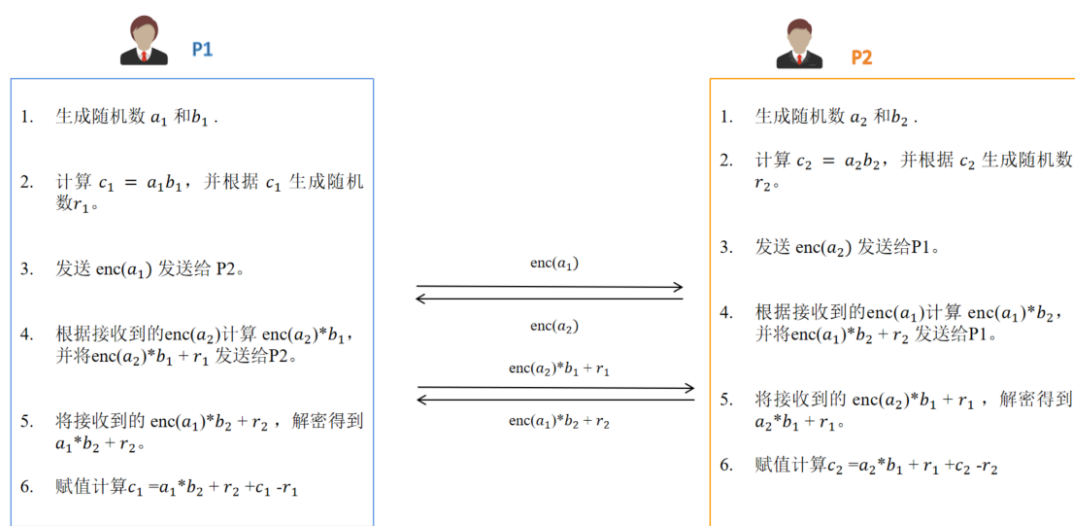


Figure 1-1 Paillier 同态加密方案构造三元组

## 1.1.2. 基于 Beaver 的 SPDZ 协议

根据构造好的三元组可以执行以下过程实现乘法门的秘密共享，记  $\alpha = a + x$  和  $\beta = b + y$ ，可以得到：

$$\begin{aligned}
 xy &= (x + a)(y + b) - (b + y)a - (a + x)b + ab \\
 &= ab - (a + x)b - (b + y)a + (x + a)(y + b) \\
 &= c - \alpha b - \beta a + \alpha \beta \\
 &= (c_1 + c_2) - \alpha(b_1 + b_2) - \beta(a_1 + a_2) + \alpha \\
 &= (c_1 - \alpha b_1 - \beta a_1 + \alpha \beta) + (c_2 - \alpha b_2 - \beta a_2)
 \end{aligned}$$

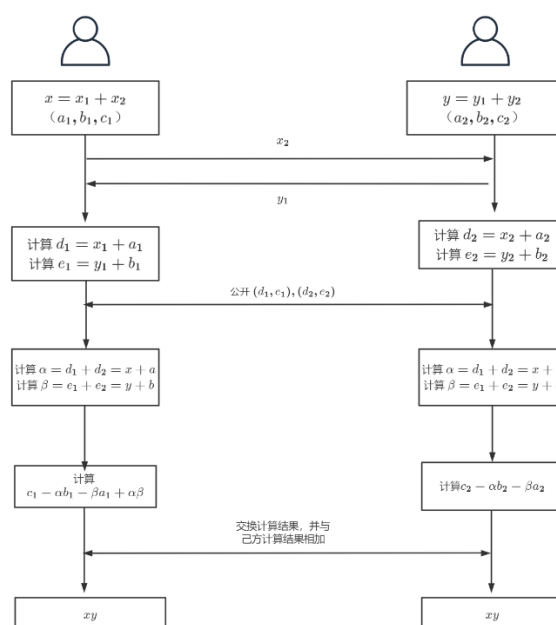


Figure 1-2 使用基于 Beaver 的 SPDZ 协议的交互过程

在现阶段需要交换得到 $a$ 和 $\beta$ 的值，就可以计算出乘法分享后的结果，在如 Figure 1-2 所示的交互过程中，因为三元组是未知的随机数，所以交互的 $d$ 和 $e$ 并不会泄露原始的 $x$ 和 $y$ ，最后交换计算结果重构可以得到 $xy$ 的值。

## 1.2. PySyft 简介

PySyft 是一个用于联合学习 (Federated Learning) 和安全多方计算 (Secure Multi-Party Computation, SMPC) 的 Python 库。它是开源的，由 OpenMined 社区开发和维护。Syft 的名称源自于 synergy (协同)，是它的简化形式。PySyft 的官方仓库为：

<https://github.com/OpenMined/PySyft>

安全多方计算 (SMPC) 作为一种在不可信环境中执行操作而不泄露数据的方式，变得越来越流行。对于机器学习模型，SMPC 将保护模型权重，同时允许多个工作节点使用自己的数据集参加训练阶段，这一过程称为联邦学习 (FL)。但是，已经表明，安全训练的模型仍然容易受到逆向工程攻击的影响，这种攻击可以直接从模型中提取有关数据集的敏感信息。另一系列被称为“差分隐私 (Differential Privacy, DP)”的方法可以解决此问题，并且可以有效保护数据。

PySyft 为每一个 PyTorch 用户提供了一个用于隐私保护深度学习的透明框架，从而可以通过直观的界面使用 FL, MPC 和 DP。下面简要介绍一下 PySyft 中较为关键的链结构：抽象张量运算的链结构。

进行变换或者将张量发送给参与方计算单元 (worker) 的行为可以表示为一个运算链，每个运算都由一个特殊的类来体现。为此，PySyft 创建了一个名为 SyftTensor 的抽象。SyftTensor 旨在表示数据的一种状态或者变换，并且可以链接在一起。链结构的头部始终有 PyTorch 张量，并使用子属性向下访问由 SyftTensor 体现的变换或状态，使用父属性向上访问由 SyftTensor 体现的变换或状态，如错误!未找到引用源。所示。

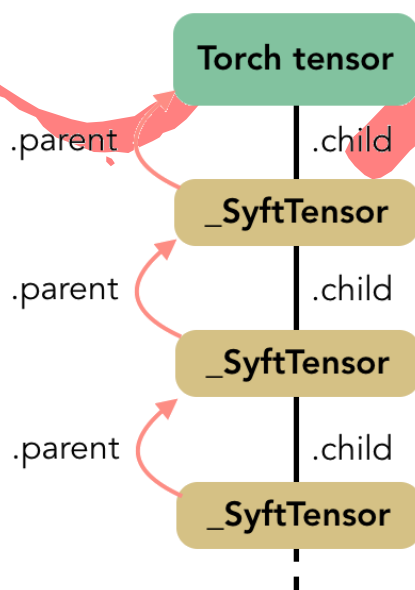


Figure 1-3 Tensor 链的一般结构

其中，`SyftTensor` 被某些具有特定作用的子类的实例替换，例如将在下面介绍的 `LocalTensor` 类。所有运算都首先被应用于 `Torch` 张量，以拥有本机 `Torch` 接口，然后将它们发送给子属性，这样它们可以通过链进行传输。

我个人理解是，假设有一个代表某些数据的 `PyTorch` 张量。然后，对这些数据应用一系列的转换，形成一个链，每个转换都被封装在一个 `SyftTensor` 中，其中原始的 `PyTorch` 张量位于链的开头（头部）。可以沿着链向下和向上导航，访问应用于数据的转换。

`SyftTensor` 有两个重要的子类：`LocalTensor` 和 `PointerTensor`，如 Figure 1-4 所示。

`LocalTensor` 在实例化 `Torch` 张量时自动创建的。它的作用是在 `Torch` 张量上执行与加载运算相对应的本机运算。例如，如果命令是 `add`，则 `LocalTensor` 将在头部张量上执行本地 `torch` 命令 `native_add`。该链有两个节点，并且是循环的，这样 `LocalTensor` 子节点指向包含数据的头节点张量，而无需重新创建子张量对象，重新创建子张量对象会降低性能。

`PointerTensor` 在当将张量发送给远程 `worker` 时创建的。在这种情况下，整个链将被发送给 `worker`，并被一个双节点链代替：张量（现在为空）和指定谁拥有数据和远程存储地址的 `PointerTensor`。这里指针没有子节点。

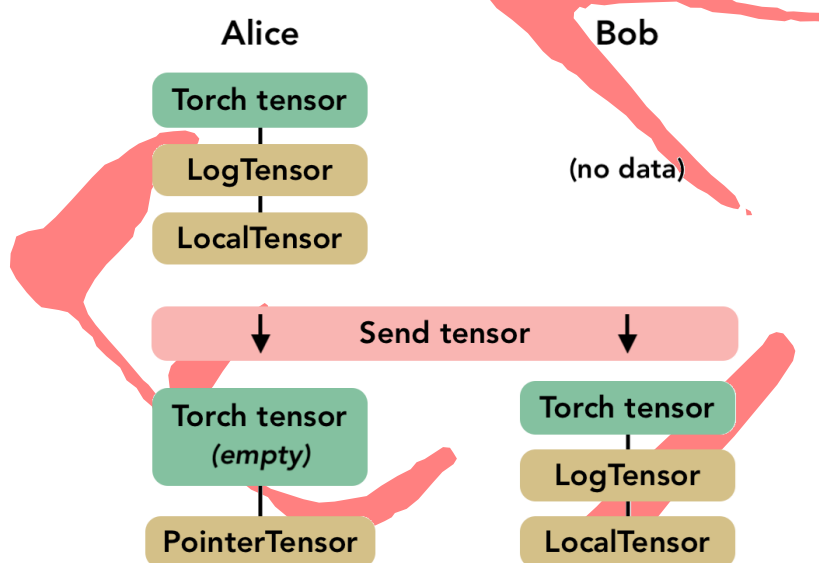


Figure 1-4 发送张量对本地和远程链的影响

为了简化调试复杂的运算链，此框架提出了 `Virtual Workers` 的概念。所有 `Virtual Workers` 都在同一台计算机上，并且不通过网络进行通信。它们只是复制命令链，并暴露与实际 `worker` 完全相同的接口，来彼此通信。

到目前为止，联邦学习环境中的网络 `worker` 在框架中有两个实现。一个建立在简单的 `network sockets` 上，另一个则支持 `Web Sockets`。`Web Sockets workers` 允许从浏览器中实例化多个 `worker`，每个 `worker` 都在自己的标签页中。这为我们提供了在实际解决不同机器上的远程 `worker` 问题之前构建联邦学习应用的另一个级别的粒度。`Web Socket workers` 也非常适合围绕基于浏览器的 `notebook` 的数据科学生态。



## 1.3. 安全多方计算框架

### 1.3.1. 建立一个 MPCTensor

1.2 介绍的元素构成了创建 `MPCTensor` 必需的基石，可以使用一系列 `PointerTensor` 来拆分和发送各个部分。同时，`PySyft` 框架中提供的 MPC 工具箱实现了 SPDZ 协议。

MPC 工具箱不仅包含基本运算（例如加法和乘法），还包含预处理工具，用于生成例如用于乘法的三元组，还包含针对神经网络的更具体的运算，包括矩阵乘法。由于 MPC 的特殊性，对卷积网络的传统元素进行了一些调整，用 `average pooling` 代替 `max pooling`，并近似了 `higher-degree sigmoid` 来代替 `relu` 作为激活函数。

SPDZ 协议假定数据是以整数形式给出的，因此 `PySyft` 在链中添加了一个 `FixedPrecisionTensor` 节点，该节点将浮点数转换为固定精度数字，然后将值编码为一个整数，并存储小数点的位置。Figure 1-5 总结了实现 SPDZ 张量的完整结构。

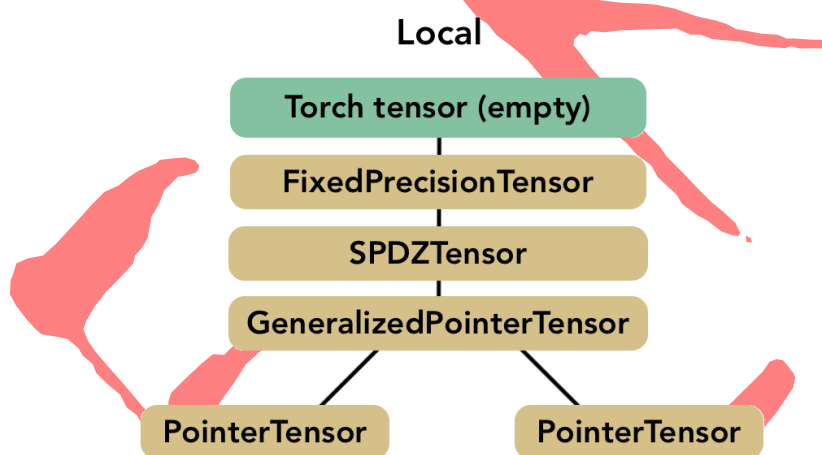


Figure 1-5 SPDZ Tensor 的链结构

与早期文献提出的 MPC 协议不同，参与者在 `PySyft` 的框架中并不平等。因为有一方是模型的所有者（称为本地 `worker`），他作为领导者，控制所有其他方（远程 `worker`）的训练过程。为了减轻处理数据时的这种集中化偏差，本地 `worker` 可以在他不拥有且看不到的数据上创建远程共享张量。

实际上，`PySyft` 希望远程 `worker` 在一般情况下能够持有一些自己的数据，例如，当医院提供医学图像用于训练模型时。然后，多个参与者都希望看到执行正确进行，这在模型推理阶段尤其重要。

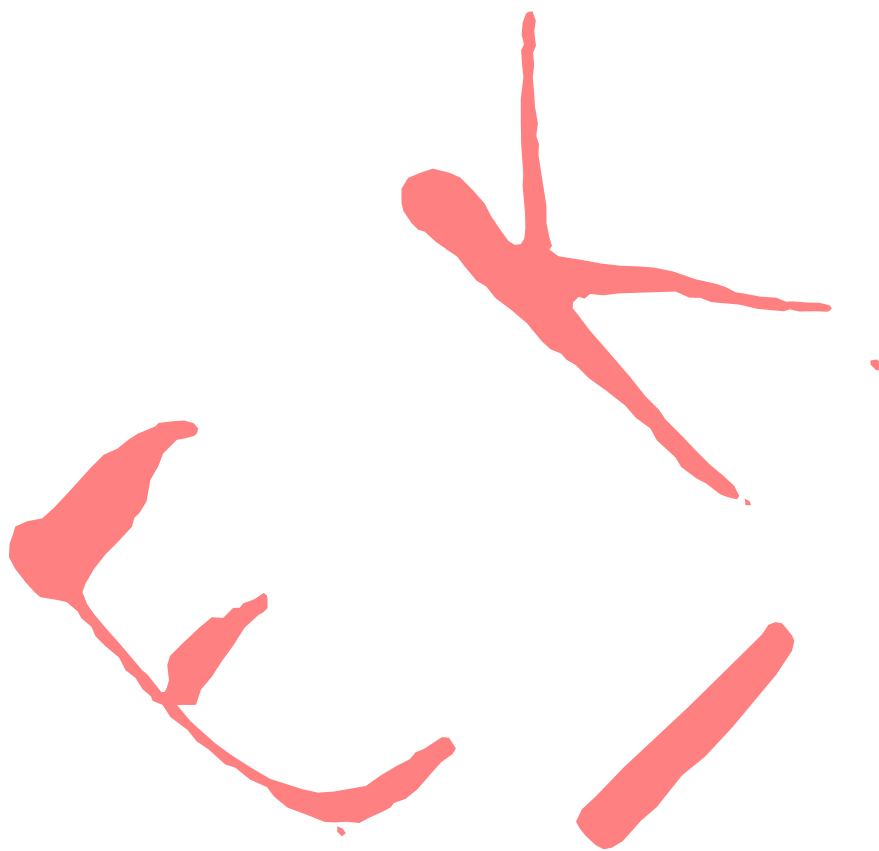
### 1.3.2. 应用差分隐私

为了给中等规模（“个位数”）隐私预算内的深度神经网络提供了一种训练方法，`PySyft` 提供了一种用于谨慎调整所需噪声的新的隐私损失估计，以及一种提高隐私训练效率的新算法。



特别是，PySyft 实施了随机梯度下降（SGD）：不是以相同的方式在数据集和 **epoch** 上进行迭代，而是由阶段构成的训练，每个阶段都包括从数据集中的 **N** 个条目中采样 **L** 个条目并用他们优化模型。此外，在直接重用了以往文献[2]的 **privacy accountant** 基础上，PySyft 实现了自己的 **sanitizer**，用于修剪梯度并添加高斯噪声。

PySyft 的框架还根据联邦学习环境提供了一些改进。首先，在进行大量抽样时，随机选择一个 **worker** 并从它自己的数据中抽样。其次，在远程 **worker** 上清理梯度，以有效确保数据私密性。这样，本地 **worker** 将获得用于更新模型的安全梯度，而无法披露有关数据集的信息。



## 2. 实验内容

### 2.1. 实验要求

全面调研一份“隐私计算”的开源框架（如 Figure 2-1 所示），运行其教程中的一个 Demo 进行代码剖析与总结。

Platform	Papers	Affiliations	Graph data and algorithms	Tabular data and algorithms	Materials
PySyft stars 8.1k	A generic framework for privacy preserving deep learning	OpenMined			Doc
FATE stars 4.3k	FATE: An Industrial Grade Platform for Collaborative Learning With Data Protection	WeBank		✓✓	Doc Doc(zh)
MindSpore Federated stars 2.9k		HUAWEI			Doc Homepage
TFF(Tensorflow-Federated) stars 1.9k	Towards Federated Learning at Scale: System Design	Google			Doc Homepage
FedML stars 1.2k	FedML: A Research Library and Benchmark for Federated Machine Learning	FedML	✓✓		Doc
Flower stars 1k	Flower: A Friendly Federated Learning Research Framework	flower.dev adap			Doc
Fedlearner stars 771		Bytedance			
LEAF stars 561	LEAF: A Benchmark for Federated Settings	CMU			
Rosetta stars 462		matrizelements			Doc Homepage
PaddleFL stars 380		Baidu			Doc
FederatedScope stars 328	FederatedScope: A Flexible Federated Learning Platform for	Alibaba DAMO Academy	✓		Doc Homepage

Figure 2-1 部分“隐私计算”的开源框架

调研的框架为教材中未详细说明的，代码剖析即对重要的代码行进行解读，调研分析越深入越好，作业报告中需要有在个人电脑中运行起来的效果图。

在以上基础上，可以对多个开源框架和多份 Demo 进行比较（非必要）。

## 2.2. 实验设计

本次实验选择了 PySyft 开源框架(<https://github.com/OpenMined/PySyft>), 运行其基本教程。

其基本教程包括四个部分:

- (1) 数据拥有者上传私有数据集;
- (2) 数据分析者提交代码(以希望在该私有数据集上运行);
- (3) 数据拥有者审核并批准该代码的执行;
- (4) 数据分析者下载/检索代码执行的结果。

以上过程是个典型的联邦学习过程。在联邦学习训练过程中, 每个参与方的原始数据不会离开该参与方, 不会被直接交换和收集。

## 2.3. 实验步骤

- (1) 下载 PySyft 官方仓库的源代码, 并配置相应环境。
- (2) 查阅 PySyft 对应发表的论文及相关文献来理解其整体架构和核心概念。
- (3) 运行 PySyft 的教程代码, 并结合参考文献和代码逻辑进行分析解读。
- (4) 撰写实验报告。

## 3. 实验分析

### 3.1. 整体示意

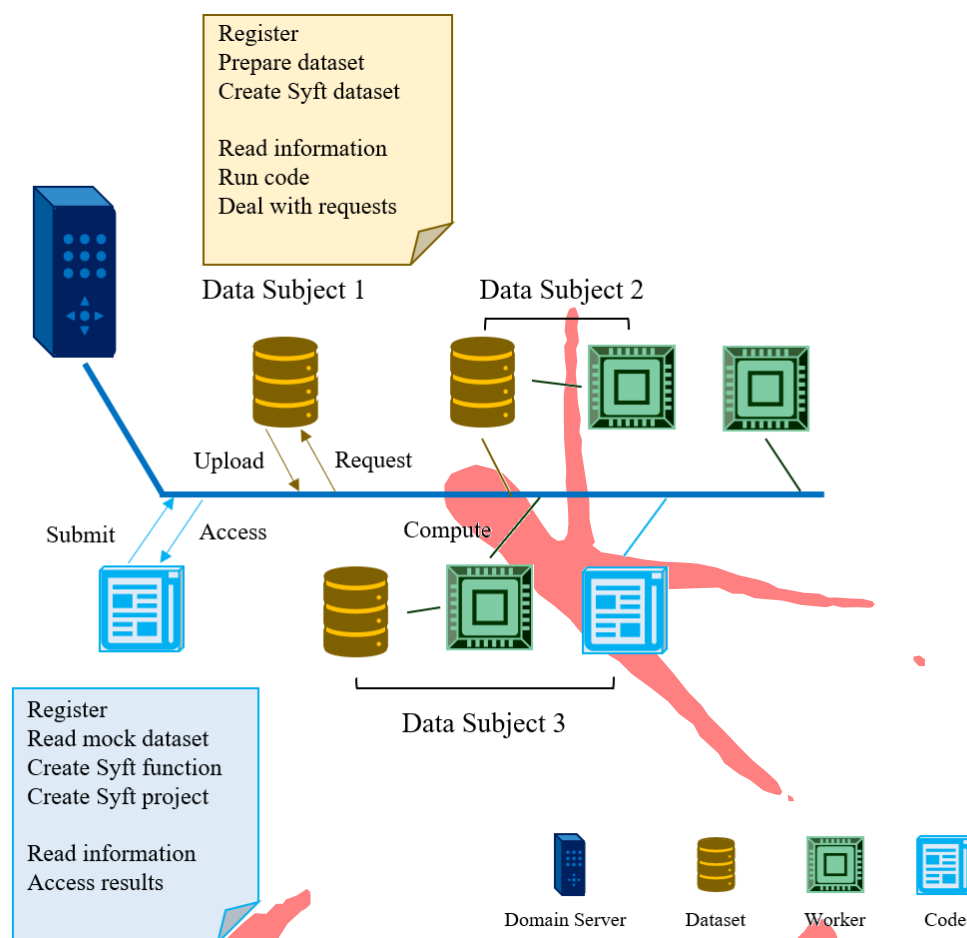


Figure 3-1 PySyft 框架的整体示意图

### 3.2. PySyft 组件

#### Domain Server

Domain Server（领域服务器）是 PySyft 中的一个重要组件，用于管理和协调不同的工作节点（workers）。Domain Server 的主要功能和用途有：

**节点注册与管理。**Domain Server 充当了 PySyft 中不同节点的注册中心。它允许工作节点注册到系统中，这些工作节点可以是本地机器上的 PySyft 节点，也可以是远程机器上运行 PySyft 的节点。

**节点发现。**Domain Server 允许工作节点发现其他已注册的节点，然后建立连接并进行通信，进而协作执行分布式任务、共享模型参数或进行联邦学习等。

**身份验证和授权。**Domain Server 负责对节点进行身份验证和授权，确保只有经过授权的节点能够参与到系统中。这有助于保障系统的安全性和隐私性。

**任务调度。**在联邦学习等场景中，Domain Server 也可以用于协调和调度任务。例如，在模型训练过程中，Domain Server 可以协调各个工作节点的贡献，确保任务的有序执行。

启动 Syft 领域服务器的核心代码如 Code 3.1 所示。这行代码指定领域服务器的名称为 "test-domain-1"，自动选择可用的服务器端口，将服务器设置为开发模式（用于本地测试和开发）。此外，在启动服务器之前，重置任何现有的服务器状态。这可以确保每次运行代码时都有一个干净的起点。

### Code 3.1: Launch Syft Domain Server

```
# Launch a fresh domain server named "test-domain-1" in dev mode on the local machine
node =
sy.orchestra.launch(name="test-domain-1", port="auto", dev_mode=True, reset=True)

# log into the node with default root credentials
domain_client = node.login(email="info@openmined.org", password="changethis")
```

如此，得到的 node 是一个对新启动的领域服务器的引用。通过这个引用，你可以与领域服务器进行交互，发送和接收数据，执行计算等。为了方便后续在该服务器上执行操作，可以使用指定的电子邮件地址和密码登录到之前启动的领域服务器，以此来生成一个领域客户端对象（domain\_client）。

### Data Subject

在 PySyft 中，Data Subject（数据主体）是指拥有数据集的个体、组织或机构，可以将数据主体视为数据的所有者。在 PySyft 中可以使用 DataSubject 类来进行操作。

处理好 DataSubject 后，不要忘记向领域服务器注册数据主体，从而将数据集分散存储在不同的工作节点上，每个工作节点都可以执行局部计算，而无需访问整个数据集。

### Syft Project

Syft Project 用于代码提交，它存储着提交者的代码和信息，提交者可以借助 Syft Project 发起提交请求。创建一个 Syft Project 的代码如 Code 3.2 所示。

### Code 3.2: Create a Syft Project

```
# Create a new project
new_project = sy.Project(
    name="My Cool UN Project",
    description="Hi, I want to calculate the trade volume in million's with my cool code.",
    members=[jane_client],
)
```

## 3.3. PySyft 概念

## Mock and Private Dataset

在 Syft 中，每个数据集都有两个变体：**Mock** 和 **Private**。

**Mock** 数据集是私有数据的模拟/虚构版本，可以被数据分析者访问和读取。**Mock** 数据集的目的是提供一个可用于开发、调试和模型设计的替代版本，而无需访问真实的私有数据。它是一个对真实数据的近似，通常用于验证算法或模型的正确性。在实际应用中，**Mock** 数据集可能由一些随机生成的数据组成，而不是直接从真实数据中提取。

**Private** 数据集是实际的私有数据，将永远不会被数据分析者访问。**Private** 数据集包含敏感信息，为了保护隐私，不允许直接访问。相反，数据分析者可以使用 **Mock** 数据集进行模型训练和其他分析工作，而不必了解或访问真实的私有数据。

这种设计允许在不暴露敏感信息的情况下进行协作研究和模型训练。数据分析者可以在 **Mock** 数据集上开发和测试算法，而不必访问实际的私有数据。一旦算法或模型在 **Mock** 数据集上表现良好，它们可以在实际的私有数据上进行验证和训练，而仍然保持数据的隐私性。这种方法有助于促进隐私保护的合作研究和联邦学习等领域的发展。

## Asset

在 Syft 中，**Dataset** 表示一个数据集，而 **Assets** 则是该数据集的组成部分。可以将 **Dataset** 理解为一个整体，而 **Assets** 则是构成这个整体的各个部分，通常是数据集的不同拆分或组件。

比如，数据贡献者的信息是一个 **Asset**，上面的 **Mock** 和 **Private** 划分是两个 **Assets**，传统的 **train**、**test** 和 **validation** 划分也是三个 **Assets**。

## Input & Output Policy

每个 Syft 函数（可能是在私有或敏感数据上操作的函数）都有相关的输入和输出策略。这些策略用于控制和验证函数执行是否符合特定的条件。

Syft 中的默认输入策略为 `sy.ExactMatch()`：确保函数只能根据数据分析者指定的确切输入执行；默认的输出策略为 `sy.OutputPolicyExecuteOnce()`：限制函数对给定输入的执行次数为一次。

该框架也允许用户基于其特定需求实现自定义策略，这种灵活性对于适应不同用例的隐私和安全机制非常重要。Syft 提供了 `@sy.syft_function_single_use()` 装饰器，以简化策略的使用。该装饰器自动将 `ExactMatch` 应用于输入和 `OutputPolicyExecuteOnce` 应用于输出。

## 3.4. 核心代码解读

### 3.4.1. Syft 函数

#### Code 3.3: Sum Trade Value

```
@sy.syft_function_single_use(trade_data=asset)
def sum_trade_value_mil(trade_data):
    import pandas as pd
    import opendp.prelude as dp
    dp.enable_features("contrib")
```

```
from opendp.measurements import make_laplace
aggregate = 0.
base_lap = dp.m.make_base_laplace(
    dp.atom_domain(T=float),
    dp.absolute_distance(T=float),
    scale=5.
)
noise = base_lap(aggregate)

df = trade_data
total = df["Trade Value (US$)"].sum()
return (float(total / 1_000_000), float(noise))
```

如 Code 3.3 所示，`@sy.syft_function_single_use` 装饰器将会自动应用 `ExactMatch` 输入策略和 `OutputPolicyExecuteOnce` 输出策略。这确保了函数只能一次性执行，而且只接受确切指定的输入。

该函数名为 `sum_trade_value_mil`，它接受参数 `trade_data`。函数内部可以导入了一些库，包括 `pandas` 用于数据处理，以及 `Syft` 框架的一些组件。

该函数使用了 `dp.make_base_laplace` 创建了一个基于拉普拉斯机制的噪声注入器。这用于在对交易数据进行总和计算时引入差分隐私的概念，以保护数据隐私。具体来说，`noise = base_lap(aggregate)` 这一行注入了噪声，其中 `base_lap` 是一个基于拉普拉斯分布的噪声生成器，`aggregate` 为零，通过这种方式引入了差分隐私。

最后，函数返回一个包含两个元素的元组，第一个元素是总交易值（以百万美元为单位），第二个元素是注入的噪声。

### 3.4.2. 设计 Policy

我们下面以 `PySyft` 默认的输出策略 `OutputPolicyExecuteOnce` 为例，分析如何设计一个 `PySyft` 策略。

Code 3.4 展示了定义 `OutputPolicyExecuteOnce` 类的继承关系，从祖先到子孙依次为：`SyftObject` -> `Policy` -> `OutputPolicy` -> `OutputPolicyExecuteCount` -> `OutputPolicyExecuteOnce`。

#### Code 3.4: Class: `OutputPolicyExecuteOnce`

```
class Policy(SyftObject):
    # version
    __canonical_name__ = "Policy"
    __version__ = SYFT_OBJECT_VERSION_1

    id: UID
    init_kwargs: Dict[Any, Any] = {}
```



```

def __init__(self, *args, **kwargs) -> None:
    if "init_kwargs" in kwargs:
        init_kwargs = kwargs["init_kwargs"]
        del kwargs["init_kwargs"]
    else:
        init_kwargs = deepcopy(kwargs)
        if "id" in init_kwargs:
            del init_kwargs["id"]
    super().__init__(init_kwargs=init_kwargs, *args, **kwargs) # noqa: B026

@classmethod
@property
def policy_code(cls) -> str:
    mro = reversed(cls.mro())
    op_code = ""
    for klass in mro:
        if "Policy" in klass.__name__:
            op_code += inspect.getsource(klass)
            op_code += "\n"
    return op_code

def public_state() -> None:
    raise NotImplementedError

@property
def valid(self) -> Union[SyftSuccess, SyftError]:
    return SyftSuccess(message="Policy is valid.")

class OutputPolicy(Policy):
    # version
    __canonical_name__ = "OutputPolicy"
    __version__ = SYFT_OBJECT_VERSION_1

    output_history: List[OutputHistory] = []
    output_kwargs: List[str] = []
    node_uid: Optional[UID]
    output_readers: List[SyftVerifyKey] = []

    def apply_output(
        self,
        context: NodeServiceContext,
        outputs: Any,
    
```

```

) -> Any:
    output_uids = filter_only_uids(outputs)
    if isinstance(output_uids, UID):
        output_uids = [output_uids]
    history = OutputHistory(
        output_time=DateTime.now(),
        outputs=output_uids,
        executing_user_verify_key=context.credentials,
    )
    self.output_history.append(history)
    return outputs

@property
def outputs(self) -> List[str]:
    return self.output_kwargs

@property
def last_output_ids(self) -> List[str]:
    return self.output_history[-1].outputs

@serializable()
class OutputPolicyExecuteCount(OutputPolicy):
    __canonical_name__ = "OutputPolicyExecuteCount"
    __version__ = SYFT_OBJECT_VERSION_1

    count: int = 0
    limit: int

    def apply_output(
        self,
        context: NodeServiceContext,
        outputs: Any,
    ) -> Optional[Any]:
        if self.count < self.limit:
            super().apply_output(context, outputs)
            self.count += 1
            return outputs
        return None

@property
def valid(self) -> Union[SyftSuccess, SyftError]:
    is_valid = self.count < self.limit

```

```

        if is_valid:
            return SyftSuccess(
                message=f"Policy is still valid. count: {self.count} < limit: {self.limit}"
            )
        return SyftError(
            message=f"Policy is no longer valid. count: {self.count} >= limit: {self.limit}"
        )

    def public_state(self) -> None:
        return {"limit": self.limit, "count": self.count}

@serializable()
class OutputPolicyExecuteOnce(OutputPolicyExecuteCount):
    __canonical_name__ = "OutputPolicyExecuteOnce"
    __version__ = SYFT_OBJECT_VERSION_1

    limit: int = 1
    
```

## Policy 类

该类继承自 `SyftObject`，这是 `PySyft` 框架中的基础类。

`id` 属性是一个唯一标识符，`init_kwargs` 是一个包含关键字参数的字典。

`__init__` 方法初始化对象，如果传递了 `"init_kwargs"` 关键字参数，则使用它，否则深拷贝其他关键字参数。

`policy_code` 是一个类方法，返回当前类及其父类中包含 `"Policy"` 的类的源代码。

`public_state` 是一个抽象方法，需要在子类中实现。

`valid` 是一个属性，返回一个表示策略是否有效的对象。

## OutputPolicy 类

该类继承自 `Policy` 类，定义了一些额外的属性，如 `output_history`、`output_kwargs`、`node_uid` 和 `output_readers`。

`apply_output` 方法接收一个上下文和输出，将输出中的唯一标识符添加到 `output_history` 中。

`outputs` 和 `last_output_ids` 是两个属性，分别返回输出的关键字参数和最后一次输出的唯一标识符。

## OutputPolicyExecuteCount 类

该类继承自 `OutputPolicy` 类，添加了 `count` 和 `limit` 两个属性，用于控制输出的执行次数。

`apply_output` 方法在执行次数未达到限制时执行父类的方法，否则返回 `None`。

`valid` 方法判断策略是否仍然有效，根据执行次数和限制进行判断。

`public_state` 方法返回对象的公共状态，包括 `limit` 和 `count`。

### OutputPolicyExecuteOnce 类

该类继承自 `OutputPolicyExecuteCount` 类，与 `OutputPolicyExecuteCount` 类相比，将 `limit` 属性固定为 1，确保输出只执行一次。

最后，`@serializable()` 装饰器用于将类转化为可序列化的对象，以便进行网络传输或存储。如此，最终的 `OutputPolicyExecuteOnce` 类便可用于定义安全的、受控制的数据输出策略，确保在特定条件下对输出进行控制和审计。

在这些继承关系中，抽象基类是 `Policy` 类，`OutputPolicy` 类是输出策略的父类（对应着输入策略），剩余类的则是具体输出策略的实现。



## 4. 实验记录

### 4.1. 数据所有者上传私有数据集

#### 4.1.1. 启动 Syft 领域服务器

```
Staging Protocol Changes...
Starting test-domain-1 server on 0.0.0.0:33600

WARNING: private key is based on node name: test-domain-1 in dev_mode. Don't run this in production.
Waiting for server to startData Migrated to latest version !!!
INFO: Started server process [41700]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:33600 (Press CTRL+C to quit)
INFO: 127.0.0.1:47612 - "GET /api/v2/metadata HTTP/1.1" 200 OK
. Done.
```

Figure 4-1 在本地启动一个名为 "test-domain-1" 的 Domain Server

```
INFO: 127.0.0.1:47900 - "GET /api/v2/metadata HTTP/1.1" 200 OK
INFO: 127.0.0.1:47900 - "GET /api/v2/metadata HTTP/1.1" 200 OK
INFO: 127.0.0.1:47900 - "POST /api/v2/login HTTP/1.1" 200 OK
INFO: 127.0.0.1:47900 - "GET /api/v2/api?verify_key=aec6ea4dfc049ceacaeecbc493167a88a200ddc367b1fa32da652444b635d;" 200 OK
INFO: 127.0.0.1:47902 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47906 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47908 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47910 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47914 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47916 - "GET /api/v2/metadata HTTP/1.1" 200 OK
INFO: 127.0.0.1:47918 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47920 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47922 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47924 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47926 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47928 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47930 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47932 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47934 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47936 - "POST /api/v2/api_call HTTP/1.1" 200 OK
INFO: 127.0.0.1:47938 - "POST /api/v2/register HTTP/1.1" 200 OK
Logged into <test-domain-1: High side Domain> as GUEST
Logged into <test-domain-1: High side Domain> as <info@openmined.org>
```

Figure 4-2 注册并生成一个领域客户端对象

```
# List the available API
domain_client.api
```

```
class SyftAPI:
    id: str = 92998bc7f40349c782be968b4fd87e27
```

Figure 4-3 生成客户端调用 Domain Server 的接口

#### 4.1.2. 添加数据主体

先生成一个数据主体的集合，用字典表示，如 Figure 4-4 所示。

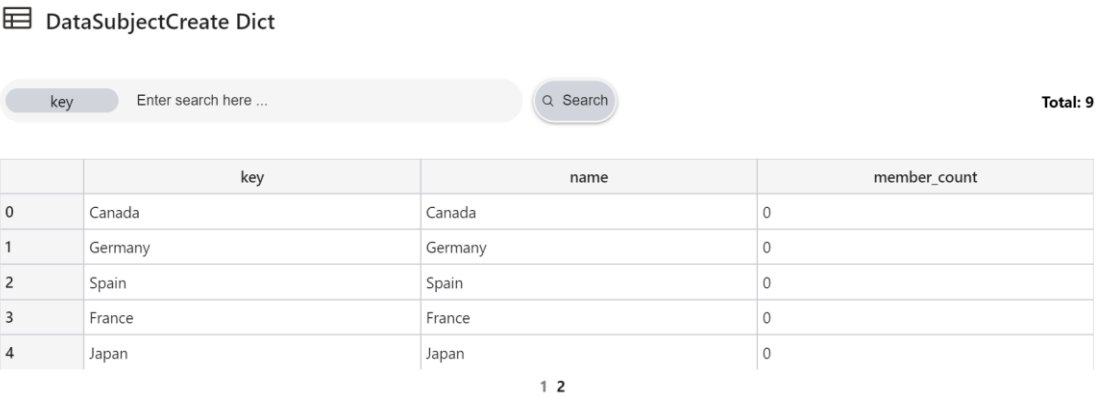


Figure 4-4 添加 Data Subject

然后向领域服务器注册这些数据主体，如 Figure 4-5 所示。

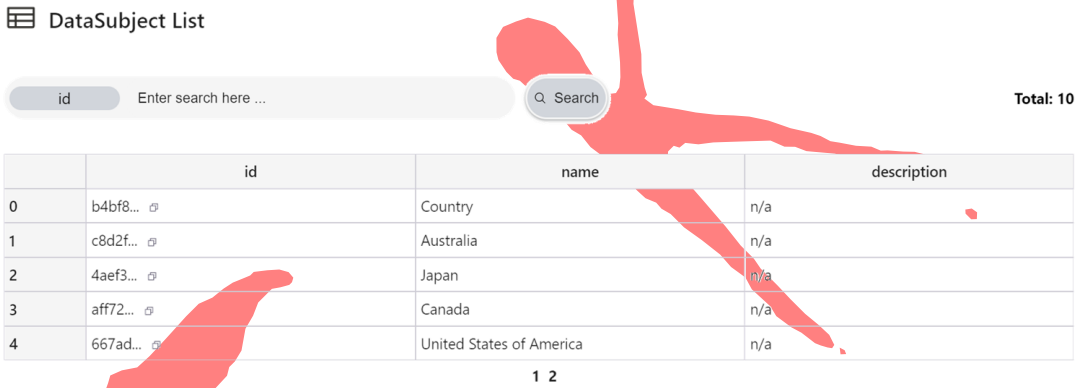


Figure 4-5 注册 Data Subject

4.1.3. 准备数据集

教程所准备的数据集是 [Canada's trade dataset](#)，如 Figure 4-6 所示。

	Classification	Year	Period	Period Desc.	Aggregate Level	Is Leaf Code	Trade Flow Code	Trade Flow	Reporter Code	Reporter	...	Partner	Partner ISO	Commodity Code
0	HS	2021	202102	February 2021	4	0	1	Imports	124	Canada	...	Other Asia, nes	NaN	6117
1	HS	2021	202102	February 2021	2	0	1	Imports	124	Canada	...	Egypt	NaN	18
2	HS	2021	202102	February 2021	2	0	1	Imports	124	Canada	...	United Kingdom	NaN	18
3	HS	2021	202102	February 2021	2	0	1	Imports	124	Canada	...	United Rep. of Tanzania	NaN	18

Figure 4-6 Canada's Trade Dataset

然后将数据集划分为 Private 和 Mock 两部分，分别如 Figure 4-7 和 Figure 4-8 所示。

```
# private data samples
ca_data = df[0:10]
ca_data
```

Python

	Classification	Year	Period	Period Desc.	Aggregate Level	Is Leaf Code	Trade Flow Code	Trade Flow	Reporter Code	Reporter	...	Partner	Partner ISO	Commodity Code	Comr
0	HS	2021	202102	February 2021	4	0	1	Imports	124	Canada	...	Other Asia, nes	NaN	6117	Clc access: ma knit
1	HS	2021	202102	February 2021	2	0	1	Imports	124	Canada	...	Egypt	NaN	18	Cocc prepar

Figure 4-7 Private Dataset

```
# Mock data samples
mock_ca_data = df[10:20]
mock_ca_data
```

Python

	Classification	Year	Period	Period Desc.	Aggregate Level	Is Leaf Code	Trade Flow Code	Trade Flow	Reporter Code	Reporter	...	Partner	Partner ISO	Commodity Code	Cor
10	HS	2021	202102	February 2021	2	0	1	Imports	124	Canada	...	Bangladesh	NaN	19	Prej c flo
11	HS	2021	202102	February 2021	2	0	1	Imports	124	Canada	...	Haiti	NaN	19	Prej c flo

Figure 4-8 Mock Dataset

4.1.4. 创建 Syft 数据集

同样地，Syft 数据集也分为 train、test 和 validation 三个部分。  
Syft 数据集可以添加贡献者的信息，如 Figure 4-9 所示。  
然后，通过 `asset` 操作，将 Syft 数据集的各个部分组装起来，形成一个 Syft 数据集，如 Figure 4-10 所示

```
dataset.contributors
```

Python

Contributor Set

id

Enter search here ...

Q Search

Total: 2

	id	name	role	email
0	a1cb1...	Madhava Jay	n/a	madhava@openmined.org
1	2827d...	Andrew Trask	n/a	andrew@openmined.org

Figure 4-9 添加 Dataset Contributor 信息



```
# This is where we add the private data (pandas df/numpy array) to the `Asset`
ctf.set_obj(ca_data)
```

```
# We must set the shape of this private data
ctf.set_shape(ca_data.shape)
```

```
# We assign the data subject for whom this data belongs to, in this
ctf.add_data_subject(canada)
```

```
# Optionally, if we don't want to add any Mock dataset
ctf.no_mock()
```

```
# We must add this Asset to our Dataset
dataset.add_asset(ctf)
```

**SyftSuccess:** Asset 'canada\_trade\_flow' added to 'Canada Trade Value' Dataset.

```
# In case we want to remove a dataset & its associated assets
dataset.remove_asset(name=ctf.name)
```

**SyftSuccess:** Asset 'Canada Trade Value' removed from 'Canada Trade Value' Dataset.

```
# Let's assign the Mock data to the Asset by calling `set_mock` method
ctf.set_mock(mock_ca_data, mock_is_real=False)
```

```
# Let's add our Asset back into our "Canada Trade Value" Dataset
dataset.add_asset(ctf)
```

**SyftSuccess:** Asset 'canada\_trade\_flow' added to 'Canada Trade Value' Dataset.

**Figure 4-10 Asset 操作创建 Syft Dataset**

### 4.1.5. 上传 Syft 数据集到领域服务器

直接使用封装好的函数上传 Syft 数据集到领域服务器, 如 Figure 4-11 所示。

```
domain_client.upload_dataset(dataset)
```


```
0%|          | 0/1 [00:00<?, ?it/s]
Uploading: canada_trade_flow
100%|██████████| 1/1 [00:00<00:00, 2.71it/s]
```

**SyftSuccess:** Dataset uploaded to 'test-domain-1'. To see the datasets uploaded by a client on this node, use command '[your\_client].datasets'

**Figure 4-11 上传 Syft Dataset 到 Domain Sever**

```
# We can list all the datasets on the Domain Server by invoking the following
datasets = domain_client.datasets.get_all()
datasets
```

Python

 Dataset Dicttuple

id Enter search here ...  Total: 1

	id	Name	Assets	Size	Url	created at
0	90538...	Canada Trade Value	1	0 (MB)	https://github.com/Open	2023-12-13 12:50:33

1

Figure 4-12 查看 Domain Sever 的 Syft Dataset

## 4.2. 数据分析者提交代码

数据分析者首先要登录到 Domain Server，如 Figure 4-13 所示。当然，在登录之前，要启动 Domain Server。

```
jane_client = guest_domain_client.login(email="jane@caltech.edu", password="abc123")
jane_client
```


✓ 1.4s Python

Logged into <test-domain-1: High side Domain> as <jane@caltech.edu>



**Welcome to test-domain-1**

URL: http://localhost:27590  
 Node Type: Domain  
 Node Side Type: High Side  
 Syft Version: 0.8.3

 This domain is run by the library PySyft to learn more about how it works visit [github.com/OpenMined/PySyft](https://github.com/OpenMined/PySyft).

**Commands to Get Started**


- <your\_client>.datasets - list datasets
- <your\_client>.code - list code
- <your\_client>.projects - list projects
- <your\_client>.code.submit? - display function signature

Figure 4-13 Data Scientist 登录到 Domain Server

### 4.2.1. 查看领域服务器的数据集

```
results = jane_client.datasets.get_all()
results
```

✓ 0.0s Python

 Dataset Dicttuple

id Enter search here ...  Total: 1

	id	Name	Assets	Size	Url	created at
0	dda85...	Canada Trade Value	1	0 (MB)	https://github.com/Open	2023-12-16 13:36:04

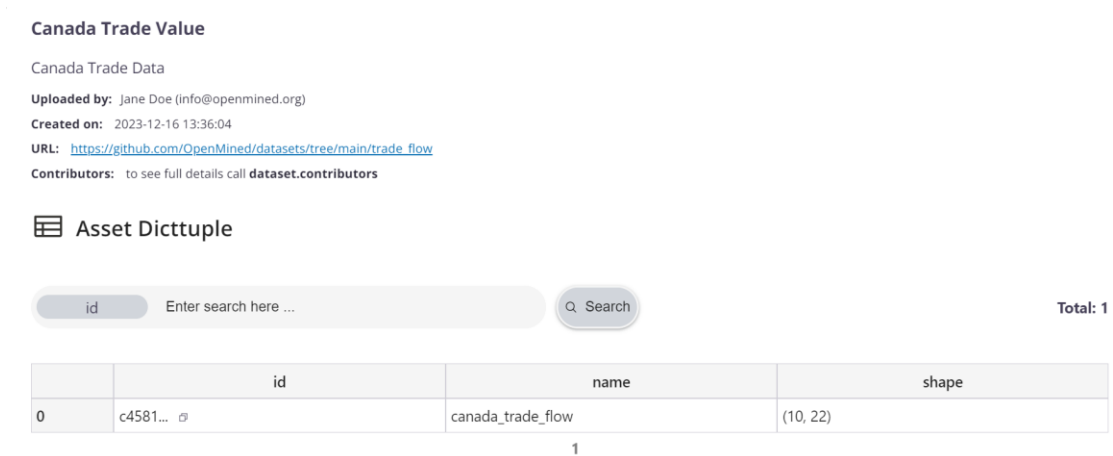
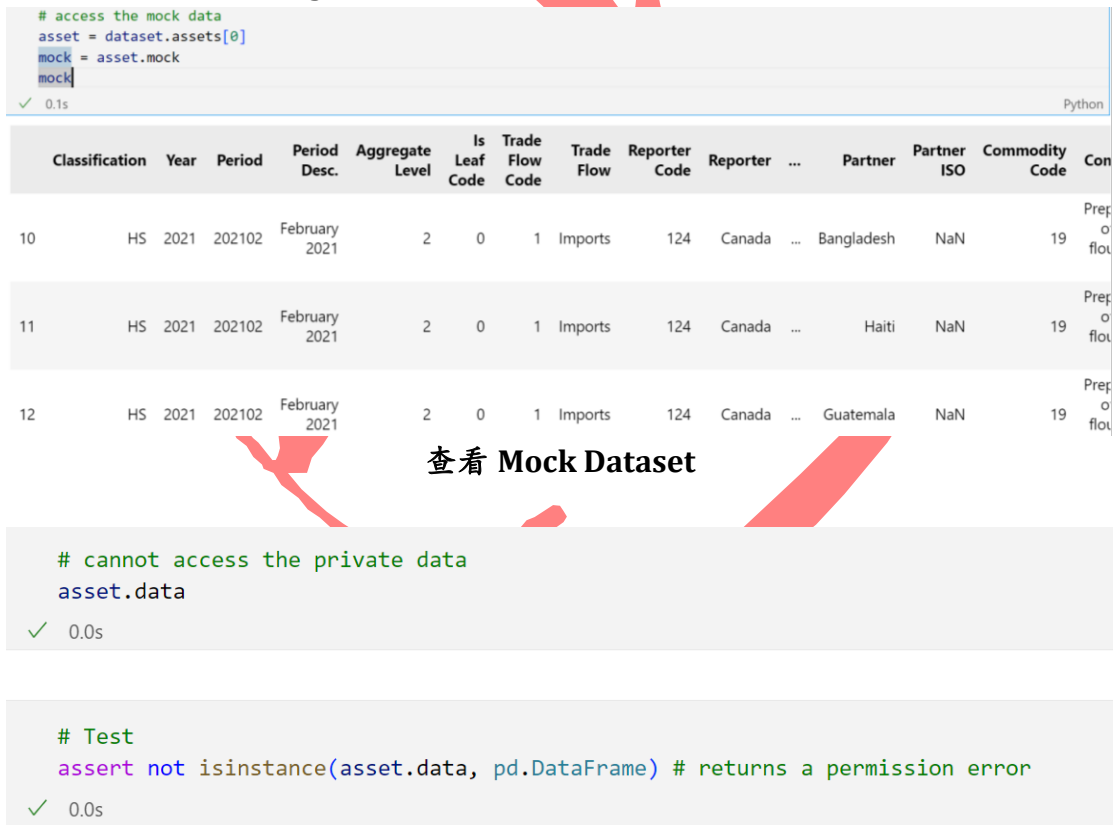


Figure 4-14 查看 Domain Server 的可用数据集

如 Figure 4-14 所示，这是在 4.1 中上传的 Syft 数据集。  
在实验分析中也提到过，作为数据分析者，只能查看 Mock 数据集而非 Private 数据集，如 Figure 4-15 所示。



不允许查看 Private Dataset

Figure 4-15 查看数据集内容

当然，我们可以直接在 Mock Dataset 上运行代码，如 Figure 4-16 所示，是一个求和的代码示例。

```
mock["Trade Value (US$)"].sum()
```

✓ 0.0s

9738381

**Figure 4-16 在 Mock Dataset 上运行代码**

同时,PySyft 也会记录对数据集的各个 Asset 的历史操作,可以通过 action\_id 查询,如 Figure 4-17 所示。

asset.id, asset.action_id	
✓	0.0s

Pythor

UID Tuple

id Enter search here ... Search Total: 2

	id
0	c4581...
1	2bec7...

**Figure 4-17 记录对数据集 Asset 的历史操作**

## 4.2.2. 创建 Syft 函数

如 Figure 4-18 所示,创建的 Syft 函数为 sum\_trade\_value\_mil。

```
print(sum_trade_value_mil.code)
```

✓ 0.0s

```
@sy.syft_function_single_use(trade_data=asset)
def sum_trade_value_mil(trade_data):
    import pandas as pd
    import opendp.prelude as dp
    dp.enable_features("contrib")
    from opendp.measurements import make_laplace
    aggregate = 0.
    base_lap = dp.m.make_base_laplace(
        dp.atom_domain(T=float),
        dp.absolute_distance(T=float),
        scale=5.
    )
    noise = base_lap(aggregate)

    df = trade_data
    total = df["Trade Value (US$)"].sum()
    return (float(total / 1_000_000), float(noise))
```

**Figure 4-18 查看 Syft 函数的具体代码**

在提交代码之前，可以在 Mock Dataset 上进行运行测试，如 Figure 4-19 所示。

```
result = sum_trade_value_mil(trade_data=mock)
result
✓ 0.1s
(9.738381, -0.8034905543662016)

assert result[0] == 9.738381
✓ 0.0s

assert isinstance(result[1], float)
✓ 0.0s

# Tests
assert len(sum_trade_value_mil.kwargs) == 1
node_identity = NodeIdentity.from_api(jane_client.api)
assert node_identity in sum_trade_value_mil.kwargs
assert "trade_data" in sum_trade_value_mil.kwargs[node_identity]
assert sum_trade_value_mil.input_policy_init_kwargs[node_identity]["trade_data"] == asset.action_id
```

Figure 4-19 在提交前测试 Syft 函数

### 4.2.3. 向领域服务器提交代码

#### My Cool UN Project

Hi, I want to calculate the trade volume in million's with my cool code.

Created by: Jane Doe (jane@caltech.edu)

```
# Add a request to submit & execute the code
new_project.create_code_request(sum_trade_value_mil, jane_client)
```

Figure 4-20 创建 Syft Project 并添加待提交代码

#### My Cool UN Project

Hi, I want to calculate the trade volume in million's with my cool code.

Created by: Jane Doe (jane@caltech.edu)



#### Request List

Total: 1

	id	Description	Requested By	Status
0	112ad...	Request to change <code>sum_trade_value_mil</code> to permission <code>RequestStatus.APPROVED</code> . Nested Requests not resolved	Jane Doe jane@caltech.edu Caltech	PENDING

Figure 4-21 发起提交请求

## 4.2.4. 运行 Syft 函数

```
result = jane_client.code.sum_trade_value_mil(trade_data=asset)
result
```

✓ 0.0s

**SyftError:** UserCodeStatus.DENIED: Function has no output policy

+ 代码 + Markdown

```
assert isinstance(result, sy.SyftError)
```

✓ 0.0s

**Figure 4-22 Data Scientist 无法直接运行 Syft 函数**

如 Figure 4-22 所示，因为数据分析者没有在数据拥有者的数据集上运行代码的权利，所以无法直接运行 Syft 函数。

## 4.3. 数据拥有者批准代码的执行

### 4.3.1. 查看信息

在批准代码的执行之前，数据拥有者须先登录到 Domain Server，可以查看数据分析者在 4.2 提交的 Syft Project，或者其他信息，如 Figure 4-23 所示。

```
class UserCode
    id: UID = ba65b4733af147fb881decd37029dc2d
    service_func_name: str = sum_trade_value_mil
    shareholders: list = ['test-domain-1']
    status: list = ['Node: test-domain-1, Status: pending']

    code:

@sy.syft_function_single_use(trade_data=asset)
def sum_trade_value_mil(trade_data):
    import pandas as pd
    import opendp.prelude as dp
    dp.enable_features("contrib")
    from opendp.measurements import make_laplace
    aggregate = 0.
    base_lap = dp.m.make_base_laplace(
        dp.atom_domain(T=float),
        dp.absolute_distance(T=float),
        scale=5.
    )
    noise = base_lap(aggregate)

    df = trade_data
    total = df["Trade Value (US$)"].sum()
    return (float(total / 1_000_000), float(noise))
```

**Code**

id

Enter search here ...

Q Search

Total: 1

	id	name	description	created by	pending requests
0	e6848...	My Cool UN Project	Hi, I want to calculate the trade volume in million's with my cool code.	jane@caltech.edu	1

Project List

canada\_trade\_flow

syft.service.dataset.dataset.MarkdownDescription

Asset ID: 83d9003a0bb742c8ba153c0baf495927

Action Object ID: 0825a64764b34de4860d9d8984bc71e8

Uploaded by: Jane Doe (info@openmined.org)

Created on: 2023-12-16 14:26:07

Data:

Classification	Year	Period	Period Desc.	Aggregate Level	Is Leaf Code	Trade Flow Code	Trade Flow	Reporter Code	Reporter
HS	2021	202102	February 2021	4	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada

Mock Data:

Classification	Year	Period	Period Desc.	Aggregate Level	Is Leaf Code	Trade Flow Code	Trade Flow	Reporter Code	Reporter
HS	2021	202102	February 2021	2	0	1	Imports	124	Canada

Asset Data

```
project.requests
```

Python

Request List

id

Enter search here ...

Q Search

Total: 1

	id	Description	Requested By	Status
0	719b6...	Request to change <code>sum_trade_value_mil</code> to permission <code>RequestStatus.APPROVED</code> . Nested Requests not resolved	Jane Doe jane@caltech.edu Caltech	PENDING

Project Request

Figure 4-23 Data Owner 查看 Domain Server 中的各种信息



### 4.3.2. 运行代码

如 Figure 4-24 所示，数据拥有者有责任去审核数据分析者提交的代码，包括确认其安全性等。如果审核通过，则可以直接运行。

```
# Let's grab the actual executable function that was submitted by the user
users_function = func.unsafe_function
```

**SyftWarning:** This code was submitted by a User and could be UNSAFE.

If the code looks safe, we can go ahead and execute it on the private dataset

```
mock_result = users_function(trade_data=mock_data)
mock_result
```

(9.738381, 0.46280272841948694)

```
real_result = users_function(trade_data=pvt_data)
real_result
```


(2.037066, 4.254573094664871)

**Figure 4-24 Data Owner 审核并运行 Data Scientist 提交的代码**


然后，可以将代码运行的结果分享给数据分析者，也是通过 Project 的 request 请求。

### 4.3.3. 处理请求

当然，除了直接运行数据分析者提供的代码，也可以否认或再次批准他的请求，分别如 Figure 4-25 和 Figure 4-26 所示。

 Request List

Total: 1

	id	Description	Requested By	Status
0	719b6... 	Request to change <b>sum_trade_value_mil</b> to permission <b>RequestStatus.APPROVED</b> Nested Requests not resolved Mutate <b>output_policy</b> to ...	Jane Doe jane@caltech.edu Caltech	REJECTED

1

**Figure 4-25 否认请求**

Request List

id Enter search here ... Search Total: 1

	id	Description	Requested By	Status
0	719b6...	Request to change <code>sum_trade_value_mil</code> to permission <code>RequestStatus.APPROVED</code> . Nested Requests not resolved Mutate <code>output_policy</code> to ...	Jane Doe jane@caltech.edu Caltech	APPROVED

Figure 4-26 再次批准请求

## 4.4. 数据分析者查看代码执行的结果

Figure 4-27 与 Figure 4-23 所示不同，在数据分析者的视角，是看不到除了 Mock Data 之外的数据的。

```
# Get the canada_trade_flow asset from the Canada Trade dataset
asset = domain_client.datasets[0].assets[0]
asset
```

**canada\_trade\_flow**

syft.service.dataset.dataset.MarkdownDescription

Asset ID: 83d9003a0bb742c8ba153c0baf495927

Action Object ID: 0825a64764b34de4860d9d8984bc71e8

Uploaded by: Jane Doe (info@openmined.org)

Created on: 2023-12-16 14:26:07

Data:

None

Mock Data:

Classification	Year	Period	Period Desc.	Aggregate Level	Is Leaf Code	Trade Flow Code	Trade Flow	Re
HS	2021	202102	February 2021	2	0	1	Imports	
HS	2021	202102	February 2021	2	0	1	Imports	

Figure 4-27 Data Scientist 查看 Domain Server 中的数据

当数据分析者拥有相关权限时，便可获得代码的执行结果，如 Figure 4-28 所示。

```
real_result = result_pointer.get()
real_result

(2.037066, 4.254573094664871)
```

```
assert real_result[0] == 2.037066
```

Figure 4-28 Data Scientist 获得代码的运行结果

因为输出策略被定义为 `OutputPolicyExecuteOnce`，所以这个 Syft 函数不能在其他输入上运行，如 Figure 4-29 所示。

```
ops = domain_client.code[-1].output_policy
ops

class OutputPolicyExecuteOnce:
    id: str = d1ea5f40af5e489d8565e4c73a70c5fe

Because the output policy is OutputPolicyExecuteOnce,

ops.valid

SyftError: Policy is no longer valid. count: 1 >= limit: 1
```

**Figure 4-29** OutputPolicyExecuteOnce 的输出策略结果

# 附录

## 附录 A：参考文献

[1] Ryffel T, Trask A, Dahl M, et al. A generic framework for privacy preserving deep learning[J]. arXiv preprint arXiv:1811.04017, 2018.

<https://arxiv.org/abs/1811.04017>

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pages 308–318, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4139-4. doi: 10.1145/2976749.2978318.

<http://doi.acm.org/10.1145/2976749.2978318>

[3] [隐私保护深度学习通用框架 \(PySyft\)](#)

[4] [隐私计算算法系列之 SPDZ 协议](#)