1. 项目总体情况

该项目为在可信执行环境SGX下进行分布式训练的XGBoost算法库，并提供了oblivious算法以保证访问模式安全。项目整体使用C++语言编写，同时有部分CUDA代码以提供GPU计算能力，以及部分python代码将项目打包安装为python库。

使用该项目前，需先将C++及CUDA部分代码编译并打包，然后使用python代码安装到python环境。安装后，可直接在python代码中调用securexgboost库进行模型的训练，示例代码如下：

**import** **securexgboost** **as** **xgb**

*# Initialize client and connect to enclave*

xgb.init\_client(user\_name="user1",

sym\_key\_file="demo/data/key\_zeros.txt",

priv\_key\_file="config/user1.pem",

cert\_file="config/user1.crt")

xgb.init\_server(enclave\_image="build/enclave/xgboost\_enclave.signed", client\_list=["user1"])

*# Remote Attestation*

xgb.attest()

*# Load the encrypted data remotely*

dtrain = xgb.DMatrix({"user1": "demo/data/agaricus.txt.train.enc"})

dtest = xgb.DMatrix({"user1": "demo/data/agaricus.txt.test.enc"})

params = {

"objective": "binary:logistic",

"gamma": "0.1",

"max\_depth": "3"

}

*# Train a model remotely*

num\_rounds = 5

booster = xgb.train(params, dtrain, num\_rounds, evals=[(dtrain, "train"), (dtest, "test")])

*# Get encrypted predictions and decrypt them locally*

predictions, num\_preds = booster.predict(dtest)

该项目提供了部分编译选项，在本次测试中重点使用的参数如下所示：

* OE\_DEBUG=1。0表示在发布模式下构建enclave，1表示在debug模式下构建。因需要测试各段代码的耗时，该选项固定为1。
* SIMULATE=ON。OFF表示使用真实enclave，ON表示模拟enclave。因真实enclave的环境尚有问题，本次测试皆使用模拟模式，该选项固定为ON。
* USE\_AVX2=OFF。是否使用AVX2指令加速。因主机不支持AVX2，固定为OFF。
* OBLIVIOUS=ON。表示是否使用oblivious操作。本次测试分别在ON与OFF条件下分别进行测试。
* LOGGING=ON。是否使用日志，固定为ON。

使用上述编译选项进行编译并安装到python环境后，使用该库训练XGBoost时，本次测试使用以下参数作为默认训练参数：

* train:6513。训练数据集大小。
* test:1611。测试数据集大小。
* tree\_method:hist。训练方法，默认为直方图算法。
* max\_depth:3。树的最大深度。
* num\_rounds:5。训练时迭代的次数。

下述测试结果除非特别说明，否则都是在oblivious设置下得到的。

1. 训练部分测试
2. 训练代码结构

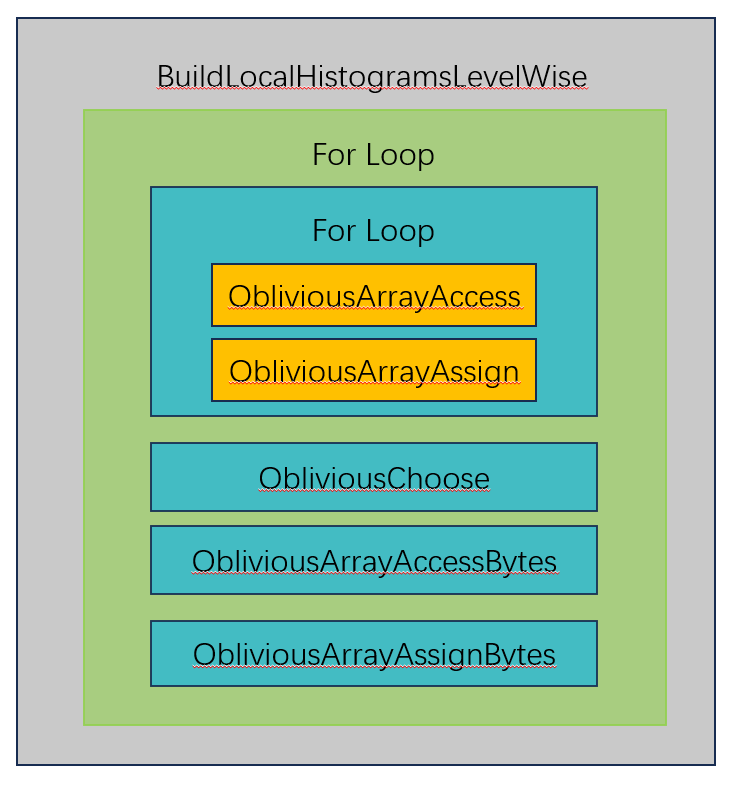
* xgb.train
  + DoBoost
    - BoostNewTrees
      * Update
        + ExpandWithDepthWidth

BuildLocalHistogramsLevelWise

* + PredictBatch

 xgb.train为库提供的python接口，它会调用num\_ rounds次DoBoost与PredictBatch，每次DoBoost进行一次迭代，构建新的树BoostNewTrees。在整个逐级的调用过程中，构建直方图操作BuildLocalHistogramsLevelWise耗时最多，占据整个训练过程xgb.train的90%以上。

其中BuildLocalHistogramsLevelWise操作的主要执行过程如下所示：



经测试，BuildLocalHistogramsLevelWise中共有五项oblivious操作，除ObliviousChoose外的四项oblivious操作占据了整个BuildLocalHistogramsLevelWise耗时的90%以上。

1. 默认参数下的测试结果



从表中可以看出，整个训练过程花费了8.7s，其中调用了BuildLocalHistogramsLevelWise操作10次，共花费8.5s，即构建直方图是训练耗时的主要来源。而对于BuildLocalHistogramsLevelWise操作，ObliviousArrayAccess、ObliviousArrayAssign两个操作总耗时4.1s，占据了一半的时间。

计时操作调用太多次，计时操作本身的耗时将无法忽略不计，而上述表格中计时操作的次数达到了百万次。为了排除计时操作造成的误差，对于调用次数达到万次以上的操作不再计时，测试结果如下：



可以看出，计时操作执行6252480次，耗时8.092s，即计时操作没执行10000次，额外增加耗时0.0129s。

为了测算oblivious操作的耗时，将ObliviousArrayAccess、ObliviousArrayAssign两个操作转化为非oblivious操作后重新测量，结果如下：



即对于一次耗时0.6358s的训练进程，ObliviousArrayAccess、ObliviousArrayAssign两个操作共占据0.329s，占据了一半的时间。

1. Max\_depth对耗时的影响

其他参数不变，将max\_depth设置为14，测试得到如下结果：



可以看出，相较于max\_depth=3时ObliviousArrayAccess、ObliviousArrayAssign为主要耗时操作，此时ObliviousArrayAccessBytes、ObliviousArrayAssignBytes成为训练耗时的主要来源。

为了排除计时操作造成的误差，首先去除了对调用次数达到万次以上的操作的计时操作，结果如下：



随后将ObliviousArrayAssignBytes、ObliviousArrayAccessBytes操作转化为非oblivious操作，得到测试结果：



即在125s的训练进程中，oblivious操作占了115s，达到了90%以上。

测量了max\_depth对总耗时的影响：



可以看出，深度每增加1，总耗时翻倍。

当设置max\_depth=15时，报错：

2023-11-20T02:53:26+0000.588959Z [(H)ERROR] tid(0x7fa36fbb4740) | :OE\_ENCLAVE\_ABORTING [/source/openenclave/host/calls.c:\_call\_enclave\_function\_impl:56]

Ecall failed: result=19 (OE\_ENCLAVE\_ABORTING)

查找错误代码的官方解释为：The operation cannot be completed because the enclave is aborting.猜测为模拟enclave内存不足。具体原因需要在真实enclave环境下测试后进行确定。在非oblivious模式下，max\_depth取值对总耗时影响可忽略不计，且没有上述错误。

1. num\_rounds对耗时的影响

其他设置不变，令num\_rounds=100：



可以看出，各部分的耗时为num\_rounds=5时的20倍。

1. 数据集大小对耗时的影响

在train:65130、test:16110，其他设置默认的条件下测试：



可以看出，数据集大小增加到10倍，则各部分耗时也增加到10倍。

取train:651300时，报错：

2023-11-20T02:53:26+0000.588959Z [(H)ERROR] tid(0x7fa36fbb4740) | :OE\_ENCLAVE\_ABORTING [/source/openenclave/host/calls.c:\_call\_enclave\_function\_impl:56]

Ecall failed: result=19 (OE\_ENCLAVE\_ABORTING)

与max\_depth过大时的错误一样，且在train:651300、非oblivious的设置下出现同样的错误。

1. 推断部分测试
2. 推断代码结构

* booster.predict
  + Leaner::Predict
    - CPUPredictor:: PredictBatch
      * CPUPredictor::PredictDMatrix
        + PredictBatchKernel

PredValue

RegTree::OGetLeafValue

booster.predict为一个python接口， RegTree::OGetLeafValue为回归树的一个方法，其根据输入特征oblivious地取树的叶子节点值。

1. 耗时主要来源

训练参数为

* train:6513
* test:1611
* tree\_method:hist
* max\_depth:10
* num\_rounds:5

得到结果：



在时长0.126s的推断过程中，OGetLeafValue操作耗时0.106s。

将OGetLeafValue操作改为非oblivious操作后结果如下：



1. 其他因素对推断的影响

经测试发现，训练次数与数据集大小对推断耗时的影响很小。Max\_depth对推断耗时的影响很大：



深度每增加1，耗时翻倍。与对训练的影响相同。在非oblivious模式下，max\_depth对推断耗时几乎无影响。

1. 结论
2. 对于训练操作，耗时主要来源于BuildLocalHistogramsLevelWise操作中的ObliviousArrayAccess、ObliviousArrayAssign与ObliviousArrayAccessBytes、ObliviousArrayAssignBytes四项oblivious操作。在整个训练过程中，ObliviousArrayAccess、ObliviousArrayAssign会执行num\_rounds\*(max\_depth+1)\*rows\*cols次，ObliviousArrayAccessBytes、ObliviousArrayAssignBytes会执行num\_rounds\*(max\_depth+1)\*rows，其中rows表示训练样本数，cols表示特征数。
3. 对于推断操作，耗时主要来源于对于每棵树执行OGetLeafValue操作。
4. Max\_depth对于oblivious模式下训练与推断操作影响极大，在非oblivious模式下几乎无影响。