USENIX Security '24 Artifact Appendix: MD-ML: Super Fast Privacy-Preserving Machine Learning for Malicious Security with a Dishonest Majority

Boshi Yuan[1], Shixuan Yang[1], Yongxiang Zhang[1], Ning Ding[1,2], Dawu Gu[1,2], and Shi-Feng Sun[1,2]

[1]Shanghai Jiao Tong University, China
[2]Shanghai Jiao Tong University (Wuxi) Blockchain Advanced Research Center
{nemoyuan2008,yangshixuan,zhang-yx7,dingning,dwgu,shifeng.sun}@sjtu.edu.cn

A Artifact Appendix

A.1 Abstract

This artifact is a C++ implementation of the protocols presented in the paper. The artifact is a standalone program that can be compiled and executed on any platform that supports C++20.

The primary contribution of this paper is the improvement of the time and communication complexities of the protocols. Therefore, the main objectives of the evaluation are to execute the program, assess its performance, and compare the findings with those presented in Section 6.

The evaluation process primarily involves compiling the source code and executing the program. The relevant statistics are automatically printed upon completion of the program's execution.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There are no security, privacy, or ethical concerns associated with this artifact.

A.2.2 How to access

https://github.com/NemoYuan2008/MD-ML/releases/tag/v0.1

A.2.3 Hardware dependencies

The experimental results in the paper were obtained with two servers. However, the artifact can be executed on a single machine.

Please make sure that the machine has at least 60 GB of free disk space to store the generated data. It is estimated that the generated data in the experiment will be around 49 GB.

A.2.4 Software dependencies

The artifact has been tested on Windows 11, Ubuntu 22.04 LTS, and macOS 14. The artifact requires the following software dependencies:

- A compiler that supports C++-20 and the integer type `__uint128_t`. This includes the following compilers:
  - GCC-10 or later,
  - Clang-10 or later,
  - Apple-Clang 13.1.6 or later.

  Note that Visual Studio (MSVC) is not supported. For Windows, we recommend MinGW-w64 with g++.[1]
- CMake 3.12 or later.[2]
- The Boost C++ libraries[3], version 1.70.0 or later. The artifact only requires the header-only `Boost::Asio` library.
- The Eigen C++ library[4], version 3.0 or later.

The following Linux softwares were used to obtain the experimental results in the paper, but they are not required to run the artifact:

- The Linux `tc` command was used to limit the bandwidth to simulate the WAN environment. The command is available on most Linux distributions.
- The Linux `iptables` command was used to measure the communication overhead. However, the communication overhead is also automatically printed by the program.

[1] https://www.mingw-w64.org/
[2] https://cmake.org/
[3] https://www.boost.org/
[4] https://eigen.tuxfamily.org/

A.2.5 Benchmarks

N/A.

A.3 Set-up

A.3.1 Installation

Step 1: Install the dependencies

On Ubuntu, the dependencies can be installed with (`$` denotes the terminal prompt):

```
$ sudo apt install build-essential cmake
    libboost-system-dev libeigen3-dev
```

On macOS, the dependencies can be installed with Homebrew:

```
$ xcode-select --install
$ brew install cmake boost eigen
```

On Windows, the dependencies can be installed manually. The following manual steps are applicable to Windows, macOS, and Linux.

- (For Windows only) Download and install MinGW-w64 from https://www.mingw-w64.org/.
- Download and install CMake from https://cmake.org/.
- Download the Boost C++ Libraries from https://www.boost.org/. Extract it somewhere and add the directory to the environment variable `PATH`. There is no need to compile the library.
- Download the Eigen C++ Library from https://eigen.tuxfamily.org/. Extract the source code, then follow the instructions in the file `INSTALL` in the extracted directory. Use "Method 2" described in the file.

Step 2: Check the compiler version

For Windows and macOS, the installation using the instructions above should provide a C++20-compatible compiler, so you can skip this step.

For Linux, the compiler installed by `apt` may be outdated and not support C++20 if you are using an older version of Linux.

- Check the version of the compiler with the following command: `g++ --version`.
- If the version is less than 10, you may need to install a newer version of the compiler, using the command `sudo apt install g++-10` (or `g++-11`).
- If the installation is successful, you need to add the option `-DCMAKE_CXX_COMPILER=g++-10` to the `cmake` command in the next step.

Step 3: Obtain and compile the artifact

Issue the following commands in the terminal:

```
$ git clone https://github.com/NemoYuan2008/MD-ML.git
$ cd MD-ML
$ mkdir build
$ cd build
$ cmake -DCMAKE_BUILD_TYPE=Release ..
$ cmake --build .
```

From now on, we will denote the build directory as `MD-ML/build`.

A.3.2 Basic Test

Make sure that you are in the `MD-ML/build` directory, then `cd ./experiments/test`. Run the following command:

```
$ ./test_fake_offline
```

The program will print nothing, but it will generate some data in the `MD-ML/fake-offline-data` directory.

Then open two terminals in the `MD-ML/build/experiments/test` directory. In the first terminal, run the following command:

```
$ ./test_party_0
```

The program will continuously print the following message, indicating that the party 0 is waiting for the party 1 to connect:

```
Failed to connect to party 1, retry after 2 seconds...
```

In the second terminal, run the following command:

```
$ ./test_party_1
```

After a few seconds, the first terminal will print messages similar to the following:

```
Spent 1 ms
Sent 64 bytes
Output: 2.25
```

and the second terminal will print a similar message without the output value 2.25.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Using the MD-ML protocol, the communication cost of AlexNet inference on the Tiny ImageNet dataset is around 1319.31 MB. This is proven by the experiment (E1) whose results are presented in Table 10 in the paper.

A.4.2 Experiments

(E1): *[AlexNet on Tiny ImageNet] [15 human-minutes + 30 compute-minutes + 50 GB disk]*
How to: The procedure to reproduce this experiment is as follows:
Preparation: cd to the directory `MD-ML/build/experiments/AlexNet-ImageNet`. Then run `./AlexNet_fake_offline` to generate the fake offline data. The generation will take around 10 minutes. The generated data will be stored in the `MD-ML/fake-offline-data` directory and will occupy around 49 GB of disk space.
Execution: Open two terminals in the same directory `MD-ML/build/experiments/AlexNet-ImageNet`. In the first terminal, run the following command:

```
$ ./AlexNet_party_0
```

In the second terminal, run the following command:

```
$ ./AlexNet_party_1
```

After the parties are connected, the program will run for less than 2 minutes.
Results: After the execution, the first terminal (running `./AlexNet_party_0`) will print the total communication overhead, which should be around 1319.31 MB.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2024/.