USENIX Security '25 Artifact Appendix: HawkEye: Statically and Accurately Profiling the Communication Cost of Models in Multi-party Learning

Wenqiang Ruan
Fudan University

Xin Lin
Fudan University

Ruisheng Zhou
Fudan University

Guopeng Lin
Fudan University

Shui Yu
University of Technology Sydney

Weili Han*
Fudan University

A Artifact Appendix

A.1 Abstract

We present HawkEye, a static model communication cost profiling framework for multi-party learning (MPL), which is an important technology for utilizing data from multiple parties with privacy preservation. Our artifact includes:

1. the accuracy evaluation of HawkEye.
2. three practical applications of HawkEye.

This artifact reproduces the tables and figures in our paper.

A.2 Description & Requirements

HawkEye enables model designers to get the accurate communication cost of PyTorch-based models in multi-party learning (MPL) frameworks without dynamically running the secure model training or inference processes on a specific MPL framework. For example, model designers can apply HawkEye to statically analyze the communication cost of different components of a CNN model constructed with PyTorch on an MPL framework (e.g., ABY3) without running the secure model training or inference process on the MPL framework. Therefore, with HawkEye, model designers can find the performance bottleneck of models efficiently and then design efficient models for different MPL frameworks.

HawkEye currently supports static model communication cost profiling on ten MPL frameworks, including CrypTFlow2, CrypTen, ABY, SPDZ-2k, ABY3, and Falcon, Delphi, Cheetah, Deep-MPC, SecretFLow-SEMI2K. Model designers can apply HawkEye to statically analyze the communication costs of PyTorch-based models on these ten MPL frameworks.

HawkEye is developed based on the compiler of MP-SPDZ that is implemented by Python. The codebase also includes the virtual machine of MP-SPDZ to reproduce experimental results in the paper.

*Corresponding author: wlhan@fudan.edu.cn

A.2.1 Security, privacy, and ethical concerns

The execution of our artifact would not cause security, privacy, or ethical risks to the machines of evaluators.

A.2.2 How to access

HawkEye can be accessed through two ways: (1) Zenodo. The URL is https://zenodo.org/records/14855032. (2) Github. The URL is https://github.com/wqruan/HawkEye. The above two repositories both include instructions for deploying HawkEye and reproducing our results.

A.2.3 Hardware dependencies

HawkEye does not require any specialized hardware. Our experiments were run on a Linux server equipped with two 32-core 2.30 GHz Intel Xeon CPUs and 512GB of RAM. The OS version is Ubuntu 20.04.

A.2.4 Software dependencies

The evaluation of our artifact needs to be performed on a Ubuntu Server that supports Python 3.8 and C++ 17. For the installation of other third-party software, we describe them in detail in the README documentation of the artifact.

A.2.5 Benchmarks

None.

A.3 Set-up

In this section, we provide information about setting up and running the artifacts.

A.3.1 Installation

We provide instructions on how to install the dependencies and necessary configuration steps in the README documen-

tation of https://github.com/wqruan/HawkEye/tree/main

A.3.2 Basic Test

After installation, you can run the command 'python compile.py -R 60 -Q CryptFlow2 -C -K LTZ,TruncPr autograd_shufflenetv2 –profiling' to check whether the installation is successful. The expected output is:

```
Conv2d
online comm size:38.87GB proportion:86.14%
online comm round:1745  proportion:67.37%
offline comm size:0.0GB proportion:0.00%
offline comm round:0.0  proportion:0.00%
Other linear
online comm size:4.15GB proportion:9.2%
online comm round:599   proportion:23.13%
offline comm size:0.0GB proportion:0.00%
offline comm round:0.0  proportion:0.00%
Linear
online comm size:43.02GB proportion:95.34%
online comm round:2344  proportion:90.5%
offline comm size:0.0GB proportion:0.00%
offline comm round:0.0  proportion:0.00%
Non_linear
online comm size:2.1GB  proportion:4.66%
online comm round:246   proportion:9.5%
offline comm size:0.0GB proportion:0.00%
offline comm round:0.0  proportion:0.00%
profiling time: 0.09888839721679688
Program requires at most:
387620574936 online communication bits
           0 offline communication bits
        2590 online round
           0 offline round
compiling time: 15.798906803131104
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): HawkEye can accurately profile the model communication cost on different MPL frameworks. This is proven by experiments (E1). The result is described in Section 5.2, Appendix B, and Appendix C of the full version of the paper and is illustrated in Tables 1, 2, 6, 7, 8, 9, and Figures 6 and 7.

(C2): HawkEye can be used to find communication bottlenecks of models on MPL frameworks with different security models, choose a proper optimizer for secure model training, and optimize the model computational graph. These are proven by experiments (E2, E3, E4). The result is described in Section 5.5 of the full version of the paper and is illustrated in Tables 4, 5, and Figure 8.

A.4.2 Experiments

(E1) Accuracy of HawkEye (80 human-minutes + 4 compute-hour + 64GB memory): This experiment evaluates the accuracy of HawkEye. We compare the communication cost profiling results outputted by HawkEye and five MPL frameworks (i.e. CrypTFlow2, CrypTen, Delphi, Cheetah, SecretFLow-SEMI2K) to show the accuracy of HawkEye.
Preparation: Build the environment as described in the following link https://github.com/wqruan/HawkEye#build-the-environment.
Execution: Use the scripts described in the following link https://github.com/wqruan/HawkEye#accuracy-of-hawkeye to run the experiments and retrieve the results. Note that the source codes of baselines are stored in other repositories. If you want to run our baselines, you can follow the instructions of REAMDE-HawkEye.md documentation in their repositories (https://github.com/wqruan/MPCFormer-HawkEye and https://github.com/yNotAVAILABLEa/Delphi-HawkEye/tree/main). The README documentation of https://github.com/wqruan/HawkEye/tree/main provides more details.
Results: The experimental results would be stored in text files. The README documentation of https://github.com/wqruan/HawkEye/tree/main provides more details. This experiment supports claim (C1).

(E2) Impact of security models (10 human-minutes + 1 compute-hour + 32GB memory): This experiment shows that HawkEye can help model designers find communication bottlenecks of models on MPL frameworks with different security models. We apply HawkEye to profile model communication cost on four MPL frameworks whose security models are different.
Preparation: Build the environment as described in the following link https://github.com/wqruan/HawkEye#build-the-environment.
Execution: Use the scripts described in the following link https://github.com/wqruan/HawkEye#impact-of-security-models to run the experiments and retrieve the results.
Results: The experimental results will be shown in a figure corresponding to Figure 8 of the full version of the paper. This experiment supports claim (C2).

(E3) Choice of optimizers (10 human-minutes + 1 compute-hour + 32GB memory): This experiment shows that HawkEye can help model designers choose a proper optimizer for secure model training. We apply HawkEye to profile the communication cost of secure model training

processes with two different optimizers.

Preparation: Build the environment as described in the following link https://github.com/wqruan/HawkEye#build-the-environment.

Execution: Use the scripts described in the following link https://github.com/wqruan/HawkEye#choice-of-optimizers to run the experiments and retrieve the results.

Results: The experimental results would be stored in text files. The README documentation of https://github.com/wqruan/HawkEye/tree/main provides more details. This experiment supports claim (C2).

(E4) Computational graph optimization (30 human-minutes + 24 compute-hour + 32GB memory): This experiment shows that HawkEye can improve the effectiveness of classical computational graph optimization in secure model inference. We combine HawkEye with TASO, a classical computational graph optimization method for deep learning models, to effectively reduce the communication overhead of secure model inference.

Preparation: Build the environment as described in the following link https://github.com/wqruan/HawkEye?tab=readme-ov-file#prepare-environment.

Execution: Use the scripts described in the following link https://github.com/wqruan/HawkEye?tab=readme-ov-file#run-experiments to run the experiments and retrieve the results.

Results: We provide a Python script to parse the experimental results. The README documentation of https://github.com/wqruan/HawkEye/tree/main provides more details. This experiment supports claim (C2).

A.5 Notes on Reusability

HawkEye can be easily reused to profile the communication costs of models on different MPL frameworks. Firstly, HawkEye can be easily deployed and run. HawkEye is purely implemented by Python and only depends on three Python libraries (i.e., numpy, matmplotlib, and setuptools) that can be installed by popular package management tools (e.g., pip). Meanwhile, HawkEye can be run by one Python command and the scripts in the folder 'HawkEye/Scripts' include many examples. Secondly, model designers can easily construct models they want to profile based on HawkEye. HawkEye provides an autograd library whose interfaces are fully consistent with PyTorch. Meanwhile, the folder 'HawkEye/Programs/Source' includes many model construction examples. Finally, HawkEye has supported ten popular MPL frameworks. In summary, model designers can easily reuse HawkEye to profile the communication costs of models on different MPL frameworks.

In addition, HawkEye can be extended to new MPL frameworks by adding the communication cost configuration of the MPL frameworks in the 'HawkEye/Compiler/cost_config.py.' Section 3.1 of the full version of the paper describes the communication cost configuration process detailedly. Meanwhile, 'HawkEye/Compiler/cost_config.py' includes many examples of MPL framework communication cost configurations.

A.6 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/usenixsec2025/.