A Artifact Appendix

A.1 Abstract

In this artifact, we will build our Cheetah framework, and to evaluate three neural networks, i.e,. ResNet50, DenseNet121, and SqueezeNet in a secure two-party computation manner. Also, we build the counterpart (i.e., SCI-HE from the CrypTFlow2's paper) for comparison. Specifically, this artifact can reproduce the performance numbers in Table 8, and Fig 10 in our paper.

To build our programs, we require a C++ toolchain including `cmake (version>=3.13)`, C++ compiler that supports C++17 (e.g., `g++>=8.0`), `make` and `git`. Also we require `OpenSSL` to be installed. To achieve the best performance, or to reproduce the performance numbers in our paper, we expect the AVX512 instructions (i.e., avx512dq and avx512ifma) are enabled.

For each neural network, we will generate two executables, one for Cheetah and the other for CrypTFlow2's counterpart. Running the executable, it will log the running time and communication cost for evaluating the neural network securely. All the logs are re-directed to file.

A.2 Artifact check-list (meta-information)

- **Algorithm:**
  Our artifact includes all the proposed algorithms in our paper. Specially, the linear protocols (Fig2, Fig4, and Fig 11) are placed in `include/gemini/cheetah/` and the non-linear protocols (Fig8 and Fig 9) are placed in `SCI/src/Millionaire/` and `SCI/src/Nonlinear/`.
- **Compilation:**
  For compilation, we provide two scripts `scripts/build-deps.sh` and `scripts/build.sh` which builds the dependencies and our implementation, respectively.
- **Binary:**
  Using our scripts, the generated binaries are placed in the `build/bin/` directory, including 6 demo.
  - `sqnet-cheetah` Run inference on SqueezeNet using Cheetah.
  - `resnet50-cheetah` Run inference on ResNet50 using Cheetah.
  - `densenet121-cheetah` Run inference on DenseNet121 using Cheetah.
  - `sqnet-SCI_HE` Run inference on SqueezeNet using SCI-HE
  - `resnet50-SCI_HE` Run inference on SqueezeNet using SCI-HE.
  - `densenet121-SCI_HE` Run inference on SqueezeNet using SCI-HE.
- **Model:**
  We provide three pretrained neural networks:
  - `pretrained/sqnet_model_scale12.inp`,
  - `pretrained/resnet50_model_scale12.inp`,
  - `pretrained/densenet121_model_scale12.inp`.
- **Run-time environment:** >=2.70 GHz CPU with more than 16GB RAM. A Linux-like OS is preferred. For instance, our timing results can be reproduced using Alibaba Cloud ecs.c7.2xlarge instances or Amazon AWS c6g.2xlarge instances.
  If to execute our artifacts on a single machine (i.e., using two processes to mimic two remote machines), we recommend the CPU supports more than 8 cores.
- **Execution:**
  We provide two scripts `scripts/run-server.sh` and `scripts/run-client.sh` to execute our demo. For example, to run an inference on SqueezeNet using Cheetah. We can run `bash scripts/run-server.sh cheetah sqnet` on one terminal and run `bash scripts/run-client.sh cheetah sqnet` on other terminal.
  Replacing the first argument `cheetah` with `SCI_HE` to run CrypTFlow2's counterpart.
- **Metrics:**
  We measure the total running time and communication cost for one inference. The one-time setup including base-ot and key-generation are NOT included. Our programs will log the running time (in seconds) and communication (in megabytes) for each layer in the neural network.
- **Output:**
  Our programs will generate a detailed log for each layer including the running time and communication. Also, on the client side, it will output the prediction label for the input image. For example, after running the script `scripts/run-client.sh cheetah sqnet`, the generated log file `cheetah-sqnet_server.log` is placed under the current directory.
- **Experiments:**
  Our artifact reproduces some empirical results in our paper, including the Cheetah and SCI-HE performance numbers in Table 8, and the top-10 values in the final prediction vectors in Figure 10.
- **How much disk space required (approximately)?:**
  About 500 megabytes, including the source codes, dependencies, built objects and pretrained models.
- **How much time is needed to prepare workflow (approximately)?:**
  It took us less than 10 minutes to build all the programs. Note that to build our programs, we need to fetch dependencies from Github.
- **How much time is needed to complete experiments (approximately)?:**
  It might take about 15–30 minutes to run all the demos.
  The three Cheetah-related demos takes less than 4 minutes to execute on LAN and AVX512 enabled. While the three SCI-HE -related demos takes more than 10 minutes to execute on LAN and AVX512 enabled.
  If AVX512 is not available, the execution time might be twice.

- **Publicly available (explicitly provide evolving version reference)?:**

  Our source codes are available in https://github.com/Alibaba-Gemini-Lab/OpenCheetah, commit hash a9b362e.

A.3 Description

A.3.1 How to access

Our source codes are available in https://github.com/Alibaba-Gemini-Lab/OpenCheetah, commit hash a9b362e.

A.3.2 Hardware dependencies

To reproduce the performance numbers in our paper, we require the CPU to support AVX512, ie., avxdq and avx512ifma instructions. Nevertheless, our programs can run without AVX512 support.

A.3.3 Software dependencies

Our programs depend on the following open-sourced libraries. Note that we provide a script `script/build-deps.sh` to fetch and build these dependencies automatically.

- emp-tool https://github.com/emp-toolkit/emp-tool
- emp-ot https://github.com/emp-toolkit/emp-ot
- Eigen https://github.com/libigl/eigen
- SEAL https://github.com/microsoft/SEAL
- zstd https://github.com/facebook/zstd
- hexl https://github.com/intel/hexl/tree/1.2.2

A.3.4 Data sets

'N/A'

A.3.5 Models

We provide three pretrained neural networks:

- `pretrained/sqnet_model_scale12.inp`,
- `pretrained/resnet50_model_scale12.inp`,
- `pretrained/densenet121_model_scale12.inp`.

These pretrained model are generated/taken from CrypTFlow2's code base https://github.com/mpc-msri/EzPC/tree/master/Athos/Networks.

A.3.6 Security, privacy, and ethical concerns

'N/A'

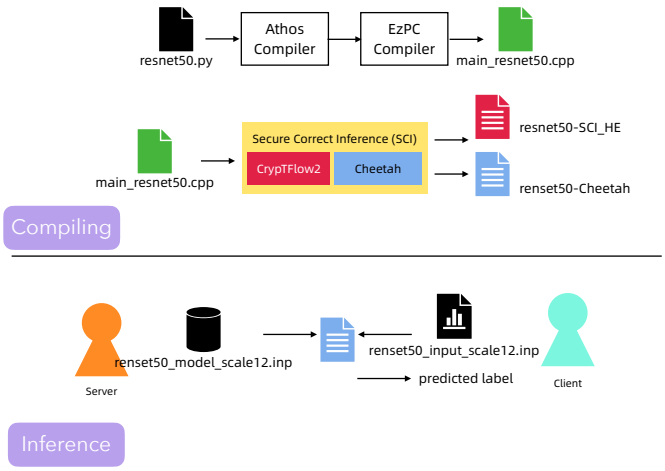

Figure 1: Workflow of Cheetah

A.4 Installation

1. Install the following requirements manuallly on your OS:
   (a) `git` We use `git` command to fetch all the source codes from Github.
   (b) `cmake` version >= 3.13. We use the `cmake` build-system to manage the source codes.
   (c) `make` To run the generated build scripts from `cmake`, we use the `make` command.
   (d) `bash` Helper scripts are written in `bash` syntax.
   (e) `openssl`. The openssl library should be installed in a 'standard' path (e.g., `/usr/include/`) so that `cmake` can find out where it is.
   (f) C++ compiler e.g. `g++` (on Linux) or `clang` (on MacOS). We require the C++ compiler to support at least C++-17. For example, `g++-8` and `clang-13`.
2. Fetch the Cheetah repo from Github via `git clone git@github.com:Alibaba-Gemini-Lab/OpenCheetah.git` Then go into the `OpenCheetah/` directory, and checkout `git checkout a9b362e` the specific version.
3. Build the dependencies via `bash scripts/build-deps.sh` This step will fetch many libraries from Github and build them, which might take a while to run.
4. Build the executables via `bash scripts/build.sh` This step will build 6 executables placed in the `build/bin/` directory.

A.5 Experiment workflow

From the high-level view, the current Cheetah implementation is an alternative implementation of the Secure and Correct Inference (SCI) Library [3]. We keep using the same interface of SCI so that we can leverage the Athos compiler [1] and the EzPC compiler [2] to convert a Python script that defines the structure of a neural network using TensorFlow to a secure two party computation C++ program that evaluates that neural work. A such compilation takes place once

for one neural network, and no trained model or input is needed during the compilation. In this artifact, we place the pre-compiled neural networks under the folder `networks/`.

For the secure inference, the server and the client run the compiled program with their private input. Also, input of server (i.e., pretrained model) and input of client (i.e., image) are also pre-processed using Athos's script. For example, floating point values are pre-processed to fixed-point values. The program will read the input from `stdin`. Also the program requires many parameters to run

- 'r' The role of player, 'r=1' indicates server and 'r=2' indicates client
- 'k' The fixed-point precision.
- 'ip' The IP address of the server.
- 'p' The port for the client's program to connect.
- 'nt' The number of threads. We can set at most 4 threads.
- 'ell' The bit length for the secret sharing, e.g,. we use 'ell=37' in our paper.

We provide helper scripts in `scripts/run-client.sh` and `scripts/run-server.sh` which hide most of the details for this parameters setting.

A.6 Evaluation and expected results

In our paper, we majorly claim two points.

1. Cheetah can evaluate deep neural network in minutes. For instance, in Table 8, we claim that Cheetah can evaluate ResNet50 within 1.5 minutes and transfer about 2.3 GB messages over LAN.
2. Our one-bit approximate truncation is effective for deep neural network inference. In § 6.5, we state that Cheetah can output almost the same prediction vector as SCI (which is bit-wise equivalent to the plaintext fixed-point computation).

By running our artifacts, we can reproduce the results in Table 8 and Figure 10.

To run our artifacts **locally**, execute as follows (take ResNet50 as the example)

1. Run `bash scripts/run-server.sh cheetah resnet50` on one terminal.
2. Run `bash scripts/run-client.sh cheetah resnet50` on the other terminal.

By replacing `cheetah` as `SCI_HE`, it will run the SCI-HE's counterpart. The other pretrained models, i.e., SqueezeNet (`sqnet`), DenseNet121 (`densenet121`) can be used by switching the second parameter.

After the computation is done, a log file is generated under the current directory, e.g., `cheetah-resnet50_client.log` on the client's machine and `cheetah-resnet50_server.log` on the server's machine. These files contain a detailed log for each layer of the neural network which can be used to validate the numbers in Table 8 and Figure 10 in our paper. The total computation time can be found in client's log file. For example in the 273-th line of `cheetah-resnet50_client.log`, it might record `Total time taken = 80719 milliseconds`. The total communication cost can be found in server's log file. For example in the 276-th line of `cheetah-resnet50_server.log`, it might record `Total comm (sent+received) = 2289.33 MiB`. The computation time might vary within 10% while the communication cost barely change much.

In addtion, we also print out the top-10 values in the final prediction vector to the last three lines in the client's log file. For our ResNet50 example, it will record

```
top-10 values from ResNet50
[13.0649084,11.7061750,10.7425666,10.4339929,9.8536843,···
predicted label=249
```

In the `SCI_HE-resnet50_client.log` (generated by running the `resnet50-SCI_HE` demo), it records

```
top-10 values from ResNet50
[13.0845959,11.7224159,10.7543676,10.4407995,9.8753999,···
predicted label=249
```

This reproduces the numbers in Figure 10 of our paper.

A.7 Experiment customization

If running on two remote machines, we first edit the `SEVER_IP` and `SERVER_PORT` variables defined in `scripts/common.sh`. The `scripts/throttle.sh` script can be used to manipulate the bandwidth (i.e., speed and ping latency). We can used this script to mimic the WAN/LAN setting within lab enviorments, e.g., running program within one machine. For example, using

```
sudo scripts/throttle.sh wan
```

on a Linux OS which will limit the local-loop interface to about 400Mbps bandwidth and 40ms ping latency. You can check the ping latency by just ping 127.0.0.1. The bandwidth can be check using extra `iperf` command.

A.8 Notes

To reproduce the timing numbers in our paper, we require the AVX512 instructions (i.e., avx512dq and avx512ifma) are supported. If AVX512 is not available, the timing numbers (both for Cheetah and SCI-HE) will be increased about 2×.

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.

References

[1] Athos. https://github.com/mpc-msri/EzPC/tree/master/Athos, June 2021.

[2] EzPC - a language for secure machine learning. https://github.com/mpc-msri/EzPC/tree/master/EzPC, June 2021.

[3] Secure and correct inference (SCI) library. https://github.com/mpc-msri/EzPC/tree/master/SCI, June 2021.