ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX Security '25 Artifact Appendix: Enabling Low-Cost Secure Computing on Untrusted In-Memory Architectures

Sahar Ghoflsaz Ghinani, Jingyao Zhang, Elaheh Sadredini
University of California, Riverside
{sghof001, jzhan502, elahehs}@ucr.edu

A Artifact Appendix

A.1 Abstract

Our artifact contains all necessary source codes and scripts for evaluating our proposed security scheme. This paper leverages Multi-Party Computation (MPC) techniques, specifically arithmetic secret sharing, and Yao's garbled circuits, to securely outsource bandwidth-intensive computation to PIM. Additionally, we employ precomputation optimizations to prevent the CPU's portion of the MPC from becoming a bottleneck. We provided all the source codes and scripts for evaluating our scheme using UPMEM, the first publicly available PIM, over four data-intensive applications: Multilayer Perceptron inference (MLP), Deep Learning Recommendation Model inference (DLRM), linear regression training, and logistic regression training. This artifact allows researchers to reproduce our results, explore this area further, and expand our work. With this artifact, researchers can regenerate the performance results for UPMEM-Precompute-C(V), UPMEM-Runtime-C(V), and UPMEM-Runtime-A(2Y)-C(V), as presented in figures 13 to 18.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our research does not compromise user privacy or safety, and all data used in our experiments is either synthetic or randomly generated. Therefore, no Personally Identifiable Information (PII), sensitive data, or human subjects were involved.

A.2.2 How to access

The code is open-source and hosted on a GitHub public repository to facilitate reuse. Additionally, it is archived on Zenodo. We encourage other researchers to explore, reproduce, and modify our code, provided they give appropriate credit. We always recommend using the latest Zenodo release via the provided concept DOI. However, we have included the link to the initial version for consistency throughout our publication.

- Zenodo initial version: zenodo.org/records/14736864.
- Zenodo concept DOI: zenodo.org/records/14736863.
- GitHub: github.com/Secure-UPMEM/SecUPMEM.git.

A.2.3 Hardware dependencies

To evaluate our method, we utilize UPMEM PIM hardware, which primarily consists of standard DDR4-2400 DIMMs integrated with DPUs. Our setup includes 20 PIM-enabled DIMMs, providing a total of 160 GB of MRAM and 2560 DPUs working in parallel at a clock frequency of 350 MHz. The host server for the UPMEM system is equipped with a 2-socket Intel Xeon Silver 4110 CPU. To accurately reproduce our results, access to the actual hardware is necessary. UPMEM's PIM data centers are accessible upon request at `https://www.upmem.com/developer.`

A.2.4 Software dependencies

All the implemented applications require the UPMEM SDK, which can be installed based on the hardware specifications and is accessible at `https://sdk.upmem.com.` For dependencies related to MLP, DLRM, logistic regression, and linear regression, please refer to our baseline implementations as our work is built upon them.

- MLP (Link)
- DLRM (Link 1, Link 2)
- Logistic and Linear Regression (Link 1, Link 2)

A.2.5 Benchmarks

Our scheme is evaluated using MLP inference, DLRM inference, logistic regression training, and linear regression training. To evaluate our implementation, we use randomly generated inputs.

A.3 Set-up

A.3.1 Installation

To run this artifact on your local device, the UPMEM SDK must first be installed, which is available at `https://sdk.upmem.com`. However, as previously mentioned, reproducing our results requires access to real hardware (20 UPMEM PIMs) rather than the simulator. Once the UPMEM SDK is installed, the artifact can be downloaded from GitHub and Zenodo.

A.3.2 Basic Test

Script *run_functionality.sh*, in the root directory, can be used to perform a simple functionality check. This file executes a basic test on all the applications sequentially and outputs execution time. To run a specific application, the *run_functionality.sh* script in the corresponding folder can be executed.

A.4 Evaluation workflow

A.4.1 Major Claims

Our major claim is as follows:

(C1): Compared to a secure CPU implementation, our framework achieves speedups of 14.66×, 9.80×, 2.64×, and 5.85× for MLP inference, DLRM inference, Linear Regression training, and Logistic Regression training, respectively. This is proven by the experiment (E1) described in Section 7.1 whose results are illustrated in Figures 14, 15, 16, 18.

A.4.2 Experiments

(E1): [30 human-minutes + 2.5 compute-hour + 32GB disk]:

How to: Our results can be reproduced by following the three steps below.

Preparation: Install the UPMEM SDK, as described in Section A.2.4, then clone our artifact.

Execution: Script *./run_reproduce.sh*, in the root directory, can be used to regenerate our results. This file executes all the applications with our configuration sequentially and outputs execution time for UPMEM-Precompute-C(V), UPMEM-Runtime-C(V), and UPMEM-Runtime-A(2Y)-C(V), as presented in Figures 13 to 18. To reproduce the results for a specific application, the *./run_reproduce.sh* script in the corresponding folder can be executed.

This script compiles and links the host and DPU source codes for a specific number of DPUs and tasklets. There is a Makefile for each application that facilitates the compilation and linking of the source codes.

Results: After running our script, the final execution time is reported. Table 1 explains the different notations used for manually determining execution times. The execution time is calculated using the formula provided below:

$$PIM\ Time = CPU - DPU + PIM\ kernel + DPU - CPU$$

$$Kernel\ time = Max[CPU\ Time, PIM\ Time]$$

$$Execution\ time = Kernel\ time + Merge + Verification$$

Table 1: List of timing notations

Timing notations	Explanation
$CPU - DPU$	CPU to DPUs transfer time
$DPU - CPU$	DPUs to CPU transfer time
$PIM\ kernel$	Kernel execution time on DPUs
$CPU\ Time$	Kernel execution time on CPU
$Kernel\ time$	The maximum of CPU Time and PIM Time
$Merge$	Merging time of CPU and PIM results
$Verification$	Verification time

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/usenixsec2025/.