AI Psychiatry: Forensic Investigation of Deep Learning Networks in Memory Images

David Oygenblik, *Georgia Institute of Technology;* Carter Yagemann, *Ohio State University;* Joseph Zhang, *University of Pennsylvania;* Arianna Mastali, *Georgia Institute of Technology;* Jeman Park, *Kyung Hee University;* Brendan Saltaformaggio, *Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity24/presentation/oygenblik

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security ’24 Artifact Appendix AI Psychiatry: Forensic Investigation of Deep Learning Networks in Memory Images

David Oygenblik[1], Carter Yagemann[2], Joseph Zhang[3], Arianna Mastali[1],
Jeman Park[4] , Brendan Saltaformaggio [1]
[1]*Georgia Institute of Technology* [2]*Ohio State University*
[3]*University of Pennsylvania* [4]*Kyung Hee University*

A Artifact Appendix

A.1 Abstract

The artifact is a code repository (with supporting documentation) for AiP, an automated memory forensics pipeline used to recover uniquely deployed ML models from system memories. AiP consists of a memory forensics application built on top of the Volatility framework to recover the ML model. AiP also includes a ML project hook, to rehost a deployed ML model into a live DL system.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact should not pose any inherent security, privacy, or ethical concerns. No preexisting data is read or transmitted. For security and ethical reasons, the artifact does not include ML backdooring tools nor does it include any active malware to perform a runtime attack on any ML models.

A.2.2 How to access

The artifact is a code repository and well-documented tutorial that can can be accessed on GitHub: `https://github.com/CyFI-Lab-Public/AiP/tree/aip-stable`

A.2.3 Hardware dependencies

While AiP does not require any specific hardware with regards to where it is deployed, AIP does require a valid OS profile (of a Linux machine) as well as a system to perform rehosting on (OS agnostic). AiP also requires a valid CUDA memory dump (for recovery of models deployed on the GPU) in addition to the main memory. For models not deployed on the GPU, a CUDA memory dump is not required.

A.2.4 Software dependencies

The preferred environment for running AiP is Ubuntu 22.04 LTS (Long Time Support). However, AiP should work with any recent version of Ubuntu. Given the fact that Ubuntu is a Debian based operating system, AiP should also work on Debian 11 and up (64 bit).

Another important component of AiP is the rest of the volatility framework (version 3) which enables AiP to identify the OS and process in which the ML model is deployed. Volatility 3 requires numerous other software dependencies to function, and the instructions can be found at `https://github.com/volatilityfoundation/volatility3/tree/v2.7.0`. The maintaining and the building of this project does not require a specific machine as long as the machine supports the Volatility 3 framework and is capable of running Python 3.7.0 or higher. The preferred environment for AiP is any Ubuntu 22.X setup or newer with a Python 3.7.X interpreter. This project also requires a Volatility 3 setup, and will not function with Volatility 2. However, AiP can, with some effort be ported to Volatility 2. Main memory dumps analyzed by AiP are collected with LiME which can be found at `https://github.com/504ensicsLabs/LiME/tree/v1.9.1`. GPU memory dumps are collected with Cuda-GDB. More information about memory aquisition can be found in the appendix of the paper.

A.2.5 Benchmarks

The primary bench mark used in the paper is the recovery and rehosting of multiple ML models (i.e. weights, layers, shapes, architecture, etc), such as those of MobileNetV2, ResNet, Bi-LSTM, and other model types used in real world applications. As these models are utilized in a variety of real world settings, and can be used during online and federated learning, AiP is evaluated on deployments of these models to verify its effectiveness. The benchmark was run on the Ubuntu 22.04 LTS operating system with AiP deployed, and the results are shown in Table 2 of the paper. We also performed an evaluation of AiPs rehosting capability in Table 3.

A.3 Set-up

A.3.1 Installation

Users should follow the Setup section of the README to deploy AiP.

A.3.2 Basic Test

Users should follow the Usage section of the README, which covers includes a step-by-step tutorial of running AiP on a SOTA ML model, namely YoloV8, which utilizes object detection to identify and bound various objects within images taken in various settings. Prior to deployment of AiP we ensured that the YoloV8 model was trained and had high accuracy.

Running AiP on a memory image containing a process deploying YoloV8 should reveal:

1. A complete summary of the types of objects within the Python process that the ML model is deployed in. Likewise, AiP will directly output the model identified (being YoloV8) and its in-memory type.

2. The types, ordering and shapes of the layers utilized in the YoloV8 model. These will be in order, and will correspond to exactly two tensors for each layer (activation and bias). The pointers to tensor buffers will be outputted, alongside each tensor's shape and number of weights.

3. Whether the tensor was on the GPU or not, and if so will output the pointer to the location in the GPU memory dump where the buffer can be found.

4. What is recovered and rehosted in AiP such that the deployed YoloV8 can be reused on the investigator's system.

The README contains step-by-step instructions for deployment and provides running examples to assist users in verifying the successful completion of each phase.

A.4 Evaluation workflow

This subsection serves to illustrate the assertions made in our paper. However, due to memory dump sizes and restrictions on testing systems, original memory dumps are not available for testing. We provide memory dumps in our accessible server for YoloV8 and MobileNetV2 implemented in PyTorch. However, upon following the instructions in the README to setup AiP and acquire memory dumps of deployed ML models, users should be able to correctly recover and rehost deployed ML models.

A.4.1 Major Claims

(C1): AiP is able to correctly recovery 30 deployed ML models out of 30 total models (100% accuracy). This is highlighted in experiment (E1) described in Section 4.2 of the paper and illustrated in Table 2.

(C2): AiP recovered all parameters of models ranging from 2.7M to 94M parameters with 100% accuracy. This is described in experiment (E2), section 4.2 of the paper, and illustrated in Table 2.

(C3): AiP correctly recovers all layers, shapes, and tensors for the deployed ML model, sucessfully filtering all tensors that are not from the ML model. This is proven by experiment (E3) described in Section 4.2 of the paper and illustrated in Table 2.

(C4): AiP recovers pointers to tensors in GPU memory for all 30 ML models. This is proven by experiment (E4) described in Section 4.1 of the paper and illustrated in Table 2.

(C5): AiP successfully rehosted all models, and all layer types, into the investigators testing system. Likewise, the models will have the same accuracy when deployed and on the investigators system. This is proven by experiment (E5) described in Section 4.3 of the paper and illustrated in Table 3.

A.4.2 Experiments

(E1): [6 human-days + 2 compute-days + 2TB storage]: Evaluate the performance of AiP in recovering deployed ML models.
Preparation: Train various ML models across multiple DL frameworks and deploy them on a Linux machine with a GPU. Collect CPU/GPU memory images during deployment.
Execution: Run AiP on memory images collected from the system when a ML model is ran on the Linux machine and collect results corresponding to the model's layers, layer shapes, and tensors.
Results: AiP should be able to recover the deployed DL model, with precision. Meaning that it has 100% accuracy on the subsequent experiments.

(E2): [6 human-days + 2 compute-days + 2TB storage]: Recover the parameters of each deployed ML model.
Preparation: Train various ML models across multiple DL frameworks and deploy them on a Linux machine with a GPU. Collect CPU/GPU memory images during deployment.
Execution: Run AiP on memory images collected from the system when a ML model is ran on the Linux machine and collect results corresponding to the parameters (weights) of the ML model.
Results: AiP should be able to recover the deployed DL model parameters, with precision. Meaning that it has 100% accuracy on the weight recovery.

(E3): [6 human-days + 2 compute-days + 2TB storage]: Execute AiP to recover all the layers, shapes, and tensors for the deployed ML model.
Preparation: Train various ML models across multiple DL frameworks and deploy them on a Linux machine with a GPU. Collect CPU/GPU memory images during deployment.

Execution: Run AiP on memory images collected from the system when a ML model is ran on the Linux machine and collect results corresponding to the the recovery of high level attributes of the ML model such as layer types, shapes, and tensors.

Results: AiP should be able to recover the deployed the high level attributes of the DL model with 100% accuracy.

(E4): [6 human-days + 2 compute-days + 2TB storage]: Run AiP to recover all tensors hosted on GPU memory for deployed ML models.

Preparation: Train various ML models across multiple DL frameworks and deploy them on a Linux machine with a GPU. Collect CPU/GPU memory images during deployment.

Execution: Run AiP on memory images collected from the system when a ML model is ran on the Linux machine and collect results corresponding to the the recovery of GPU pointers/buffers of tensors corresponding to the layers of the ML model when the model is deployed on the GPU.

Results: AiP should be able to recover the tensors placed in GPU memory.

(E5): Correctly rehost the layers and parameters recovered from the deployed ML model into an investigator's system.

Preparation: Utilizing the output of AiPs recovery (i.e. recovered tensors, layers, tensor buffers, etc), the investigator can begin AiP's rehosting process.

Execution: Run the rehosting script (following the README) to get the model in the live environment to utilize what was recovered from the deployed ML model.

Results: AiP correctly recovers and rehosts all of the layers and parameters (weights) of the ML model in an investigative environment.

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/.