Decompiling x86 Deep Neural Network Executables
Zhibo Liu, Yuanyuan Yuan, and Shuai Wang, The Hong Kong University of Science and Technology; Xiaofei Xie, Singapore Management University; Lei Ma, University of Alberta
https://www.usenix.org/conference/usenixsecurity23/presentation/liu-zhibo

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.
August 9–11, 2023 • Anaheim, CA, USA

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.
A Artifact Appendix

A.1 Abstract

We provide source code of BTD and data used in our experiments. Our artifact is publicly available at https://github.com/monkbai/DNN-decompiler/tree/b4f64783846b85cac4b0eb6c7a5595535cc858d3 with detailed documents. In the evaluation, user is able to use BTD to decompile 63 provided DNN executables into their original DNN model specifications, including ① DNN operators and their topological connectivity, ② dimensions of each DNN operator, and ③ parameters of each DNN operator, such as weights and biases, in json format.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our artifact does not rise any ethical concerns. The experiments will not cause any risk for evaluators’ machines security or data privacy.

A.2.2 How to access

The artifact is publicly available at https://github.com/monkbai/DNN-decompiler/tree/b4f64783846b85cac4b0eb6c7a5595535cc858d3.

A.2.3 Hardware dependencies

We ran our evaluation experiments on a server equipped with Intel Xeon CPU E5-2683, 256GB RAM, and an Nvidia GeForce RTX 2080 GPU. Logging and filtering all traces for all DNN executables in the evaluation takes more than a week and consumes nearly 1TB disk storage. To ease the AE committee to review, we omit the trace logging process and provide the filtered traces in the docker image and evaluation data. The trace logger and filter are provided in MyPinTool and the trace_filter.py script. Without logging and filtering, the whole evaluation takes roughly 24 hours and requires less than 120GB of disk space. Besides, the symbolic execution may consume a lot of memory resources, so please make sure that the machine on which the experiment is run has sufficient memory.

A.2.4 Software dependencies

BTD relies on IDA Pro (version 7.5) for disassembly, and because IDA is commercial software, we do not provide it in this repo; instead, in order to reduce the workload of AE reviewers, we provide the disassembly results directly as input for BTD. The scripts used to disassemble DNN executable into assembly functions with IDA are presented in our artifact. IDA Pro is not indispensable; any other full-fledged disassembly tool can be used to replace IDA.

A.2.5 Benchmarks

Table 1: Compilers evaluated in our study.

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Publication</th>
<th>Developer</th>
<th>Version (git commit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVM</td>
<td>OSDI ’18</td>
<td>Amazon</td>
<td>v0.7.0, v0.8.0, v0.9.dev</td>
</tr>
<tr>
<td>Glow</td>
<td>arXiv</td>
<td>Facebook</td>
<td>2020 (0fa28b9f9c79d2)</td>
</tr>
<tr>
<td>NNFusion</td>
<td>OSDI ’20</td>
<td>Microsoft</td>
<td>v0.2, v0.3</td>
</tr>
</tbody>
</table>

Our evaluation covers above 7 models compiled with 9 different compiler options, including Glow-2020, Glow-2021, Glow-2022, TVM-v0.7 (O0 and O3), TVM-v0.8 (O0 and O3), TVM-v0.9.dev (O0 and O3), in total 63 DNN executables. NNFusion-emitted executables are easier to decompile since they contain wrapper functions to invoke target operator implementations in kernel libraries (see our paper for more detailed discussion). Thus, in this evaluation we only focus on decompiling executables compiled by TVM and Glow.
Table 2: Statistics of DNN models and their compiled executables evaluated in our study.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Parameters</th>
<th>#Operators</th>
<th>TVM -O0</th>
<th>TVM -O3</th>
<th>Glow -O3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet18</td>
<td>11,703,912</td>
<td>69</td>
<td>49,762</td>
<td>61,002</td>
<td>11,108</td>
</tr>
<tr>
<td>VGG16</td>
<td>138,357,544</td>
<td>41</td>
<td>40,205</td>
<td>41,750</td>
<td>365,992</td>
</tr>
<tr>
<td>FastText</td>
<td>2,500,101</td>
<td>3</td>
<td>9,867</td>
<td>7,477</td>
<td>405</td>
</tr>
<tr>
<td>Inception</td>
<td>6,998,552</td>
<td>105</td>
<td>121,481</td>
<td>74,992</td>
<td>30,452</td>
</tr>
<tr>
<td>Shufflenet</td>
<td>2,294,784</td>
<td>152</td>
<td>56,147</td>
<td>34,637</td>
<td>33,537</td>
</tr>
<tr>
<td>Mobilenet</td>
<td>3,487,816</td>
<td>89</td>
<td>69,903</td>
<td>46,214</td>
<td>37,331</td>
</tr>
<tr>
<td>Efficientnet</td>
<td>12,966,032</td>
<td>216</td>
<td>89,772</td>
<td>49,285</td>
<td>13,749</td>
</tr>
</tbody>
</table>

A.3 Set-up

A.3.1 Installation

Download the packed docker image, then run the command below to unpack the .tar file into a docker image.

```
cat BTD-artifact.tar | docker import - btd
```

Create a container with the docker image.

```
docker run -dit --name BTD-AE btd /bin/bash
```

Open a bash in the container:

```
docker exec -it BTD-AE /bin/bash
```

cd /home

BTD can also be installed from source code, the detailed instructions are listed in our artifact.

A.3.2 Basic Test

To run the evaluation of operator inference:

```
cd DNN-decompiler
git pull
./op_infer_eval.sh
```

Inference results are written in the output directory. The output would be in format: Compiler Option-Model-Operator Name/Type Pred: output. For example, the output below indicates that a libjit_fc_f (Fully-Connected, FC) operator in the vgg16 model compiled with Glow_2021 is correctly inferred as matmul (Matrix Multiplication).

```
GLOW_2021-vgg16-libjit_fc_f Pred: matmul
GLOW_2021-vgg16-libjit_fc_f Label: matmul
```

To run the evaluation of decompilation and rebuild:

```
cd DNN-decompiler
git pull
./decompile_eval.sh
```

This experiment will decompile and rebuild all 63 DNN executables. It takes 24 hours to finish all experiments. The output of rebuilt models and original DNN executables will be printed on screen (see example in Decompilation Correctness below). Corresponding decompilation outputs will be stored in the evaluation directory.

After executing decompile_eval.sh, for each directory in evaluation, a topo_list.json containing the network topology (1), a new_meta_data.json containing dimensions information (2), and a series of func_id.weights/biases_id.json containing all parameters of the decompiled DNN model (3) will be generated.

Each item in topo_list.json will be: ['node id', 'func_id.txt', 'operator type', [input addresses], 'output address', [input node ids], occurrence index]

Each item in new_meta_data.json will be: ['<func_id>.txt', [operator dimensions], 'operator entry address (in executable)', 'operator type', with_parameter, stride (if exists), padding (if exists)].

Examples can be found in README.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): BTD is able to decompile all 63 DNN executables into model specifications that are (near) identical with input models. The decompiled model specifications can be used to rebuild new models that have identical output (with minor precision loss) as the output of original DNN executables.

A.4.2 Experiments

After decompilation experiments, all DNN model are rebuild with decompiled model structures and extracted parameters (stored in .json format). decompile_eval.sh will run each rebuilt model (implemented in pytorch) and the original DNN executable with the example image in binary format as input. The output would be like this:

```
- vgg16_tvm_v09_03
  Result: 282
  Confidence: 9.341153
```
In the above example, both rebuilt model and DNN executable output result as 282 (see 1000 classes of ImageNet), and the confidence scores are 9.341153 and 9.341150, respectively. While the confidence scores (or max values) are slightly inconsistent, we interpret that such inconsistency is caused by the floating-point precision loss between pytorch model and DNN executable, i.e., the decompilation is still correct.

\textbf{(E1): [Decompilation Correctness]} \[10 \text{ human-minutes} + 24 \text{ compute-hour} + 120 \text{GB disk}]: \text{as described above.}

\textbf{How to:} \ As described in \textbf{A.3.2 Basic Test}.  

\textbf{Results:} \ The predicted label output by the original DNN executable and the rebuilt model should be identical.

\section{A.5 Version}

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at \url{https://secartifacts.github.io/usenixsec2023/}.