GoldenEye

Release 1.0

Jun 29, 2022

CONTENTS

CHAPTER

ONE

OVERVIEW

Goldeneye is a functional simulator of different number systems with fault injection capabilities. It is built on top of the Pytorch deep learning framework and the PytorchFi fault injection library. Goldeneye can be used either for conducting fault injection campaigns or the general study of the reliability and accuracy of various number systems. This documentation gives an overview of the different use-cases of Goldeneye and how to customize it to your own number systems and models.

CHAPTER

TWO

GETTING STARTED

This documentation provides an overview of the features of Goldeneye including its usage and how to customize it to your specific use-cases.

CHAPTER

THREE

CONTENTS

3.1 Setup

Contents

3.1.1 Installation

Ubuntu with sudo privileges

1. Recursively clone the goldeneye repository.

```
git clone --recurse-submodules git@github.com:ma3mool/goldeneye.git
```

2. Download ninja-build which is needed for qtorch.

```
sudo apt install ninja-build
```

3. Download the other project dependencies. Please make sure you are inside the goldeneye folder when applying this command.

```
pip install -r requirements.txt
```

4. Setup environment variable (replace with the directory where the imagenet dataset is downloaded).

```
ML_DATASETS=/dir/to/imagenet/
```

Docker

1. Recursively clone the goldeneye repository.

```
git clone --recurse-submodules git@github.com:ma3mool/goldeneye.git
```

2. Pull the goldeneye docker image and rename it to simply the next steps

```
docker pull goldeneyetool/goldeneye:latest
docker image tag goldeneyetool/goldeneye goldeneye
```

3. Within the goldeneye folder, run the shell on the pulled docker image. Make sure to replace [/path/to/imagenet] with the actual path to your downloaded imagenet dataset.

```
cd goldeneye
docker run -ti
    --mount type=bind,source=`pwd`/src/,target=/src
    --mount type=bind,source=`pwd`/val/,target=/val
    --mount type=bind,source=`pwd`/scripts/,target=/scripts
    --mount type=bind,source=[/path/to/imagenet],target=/datasets/imagenet
    goldeneye
```

3.1.2 Code Overview

Scripts Folder (scripts)

The **scripts** folder includes wrappers around the goldeneye framework to simplify its use.

Source Folder (src)

The **src** folder contains all of the goldeneye core logic such as number system implementation and error injection routines.

Validation Folder (val)

The **val** folder is used for unit-testing the code. You can run it using pytest to check that the installation process was successful.

3.2 Usage

Contents

- *Domain Space Exploration*

3.2.1 How to Use

To add custom models and/or pretrained weights: 1. add your model to *src/othermodels/* 2. add pt files to *src/othermodels/state_dicts* 3. modify *getNetwork()* in *src/util.py* to correspond to the dataset and model name. 4. also add the appropriate *import* in */src/util.py*

Your model.py file in *src/othermodels/* should have a few general attributes: 1. A codeblock similar to easily find your model versions

```
__all__ = [
    "baseline",
    "v1",
    "v2",
]
```

2. A separate function for each name in Step 1, which instantiates your model and extracts the correct model parameters from `/src/othermodels/state_dicts/`. As an example, check out: https://github.com/huyvnphan/PyTorch_CIFAR10/blob/master/cifar10_models/resnet.py

3.2.2 Model accuracy

In order to evaluate the accuracy of your model, you can directly use the accuracy_profile script, which will run the preprocess, profile, and split_data python scripts from src/

```
./scripts/accuracy_profile.sh NETWORK BATCH FORMAT BITWIDTH RADIX
```

3.2.3 Model Resiliency

In order to evaluate the resiliency of your model, you should run the end_to_end script, which will run all the python scripts from src/ in order (preprocess, profile, split_data, injections and postprocess)

```
./scripts/end_to_end.sh NETWORK DATASET BATCH FORMAT BITWIDTH RADIX INJECTIONS_LOC
```

3.2.4 Domain Space Exploration

In order to conduct Domain Space Exploration, you should run the script sweep.sh which will run the python file sweep_num_formats.py

```
./scripts/sweep.sh
```

3.3 Customization

Contents

3.3.1 Adding custom number systems in Python

Adding custom number systems in Goldeneye is straightforward. You have to implement a sub-class for your number system with 4 necessary methods and one optional method (real_to_format_tensor_meta).

1. Open "src/num_sys_class.py".
2. Implement a class that inherits from "_number_sys" with its __init__ method.
3. Implement the "real_to_format" method which turns a real number into a bitstring denoting the representation of that number in your custom number system.
4. Implement the real_to_format_tensor method accepts a real tensor and returns a an equivalent tensor in the simulated custom number format.
5. Implement "format_to_real" which accepts a bitstring in the custom number system and returns a real number
6. [OPTIONAL] Implement the real_to_format_tensor_meta method which accepts a real tensor and returns an equivalent tensor in the simulated custom number system. This method is specific to converting from real to the number system's metadata. This can only be useful in number systems where there is some metadata that is stored separate from the individual numbers (e.g. common exponent in block floating point).

Please refer to the pre-implemented number systems under "src/num_sys_class.py" as guiding examples (e.g. block_fp and adaptive_float).

3.3.2 Adding custom number systems in C++

The class framework for building a number system in C++ is identical to that in Python. We will make use of PyTorch's library to allow us to run our C++ code inside a Python function. At the top of our .cpp file, we can simply type `#include <torch/extension.h>`, which will give us all the necessary libraries we need to create our functions.

1. Similar to the Python implementation, initialize a sub-class with the 4 necessary methods.
2. Write the corresponding functions in C++ in "src/num_sys.cpp". You should also add the function prototypes to the header file, "src/num_sys.h", in case you would like to do some testing in a separate file in C++.
3. At the end of the .cpp file, you will need to add a pybind module, similar to the one below. This will allow you to call your C++ functions inside the Python code. Here you can replace "func_name_py" with the name you would like to call your function by in Python, "func_name_cpp" with the name of the C++ function in "src/num_sys.cpp", and "Description" with a short description of your function in plaintext.

```
PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
    m.def("func_name_py", &func_name_cpp, "Description");
}
```

4. Finally, you will need to import these functions into your .py file. We will use just-in-time compilation, since it avoids the use of a "setup.py" file. In order to do so, we need to include `from torch.utils.cpp_extension import load` in our "src/num_sys_class.py" file. We then include the following where where "module_name" can be replaced with the module name you want to use and "directory_to_cpp_file" is the path from "src/" to your .cpp file.

```
current_path = os.path.dirname(os.path.realpath(__file__))
module_name = load(
    name="module_name",
    sources=[
        os.path.join(current_path, "directory_to_cpp_file"),
    ]
)
```

Adaptive Float and Block Float were both implemented using C++ under "src/num_sys.cpp", so you can refer to those functions as needed.

CHAPTER

FOUR

ABOUT THE PROJECT

This project was developed by the Harvard Architecture, Circuits and Compilers research group.

Goldeneye is an open-source functional simulator of different number systems with fault injection capabilities. It is built around the pytorch ecosystem and the Pytorchfi fault injection library.

4.1 Contributors

- Abdulrahman Mahmoud
- Thierry Tambe
- Tarek Aloui
- David Brooks
- Gu-Yeon Wei
- Josh Park

CHAPTER

FIVE

CITATION

If you use or reference Goldeneye, please cite:

```
@INPROCEEDINGS{GoldeneyeMahmoudTambeDSN2022,
author={A. {Mahmoud} and T. {Tambe} and T. {Aloui} and D. {Brooks} and G. {Wei}},
booktitle={2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and␣
↪Networks (DSN)},
title={GoldenEye:  A Platform for Evaluating Emerging Data Formats in DNN Accelerators},
year={2022},
}
```

CHAPTER

SIX

LICENSE