****

File Structure

****

```
/home/pldi2023

├── README.md                 - this file.
├── requirements.txt          - dependencies to run incremental verifier
IVAN.
├── nnverify                  - Code for the incremental verifier.
   analyzer.py                - code for individual property
verification.
   ├── bnb                    - the branch and bound-based verification.
       ├──bnb.py              - branch and bound verification algorithm.
       ├──branch.py           - code for deciding the branching
strategy.
       ├──proof_tree.py       - (Core contribution) specification tree
code (section 2.2).
   ├── domains                - verification methods (DeepZ, lp-based).
       ├── deepz.py           - code for the DeepZ verifier.
       ├── lp.py              - code for the linear programming-based
verifier.
   ├── proof_transfer         - code for incremental verification.
       ├──approximate.py      - code for computing modified network.
       ├──proof_transfer.py   - (Core contribution) incremental
verification on (Algorithm 4).
       ├──template.py         - code for storing specification tree.
       ├──param_tune.py       - tuning the parameters(α and θ)(Algorithm
4).
   ├── tests                  - code for reported experiments.
       ├──test_pldi.py        - code for experiments reported in the
paper.
```

****

Reproducing Testing Experiments

****

Step 1: Installing Gurobi

1. GUROBI installation instructions can be found at https://www.gurobi.com/documentation/9.5/quickstart_linux/software_installation_guid.html

   For Linux-based systems the installation steps are:

   Install Gurobi:

   ```
   wget https://packages.gurobi.com/9.1/gurobi9.1.2_linux64.tar.gz
   ```

```
tar -xvf gurobi9.1.2_linux64.tar.gz
cd gurobi912/linux64/src/build
sed -ie 's/^C++FLAGS =.*$/& -fPIC/' Makefile
make
cp libgurobi_c++.a ../../lib/
cd ../../
```

`cp lib/libgurobi91.so /usr/local/lib` -> (You may need to use **sudo** command for this)

```
python3 setup.py install
cd ../../
```

Update environment variables:

i) Run following export commands in command prompt/terminal (these environment values are only valid for the current session)

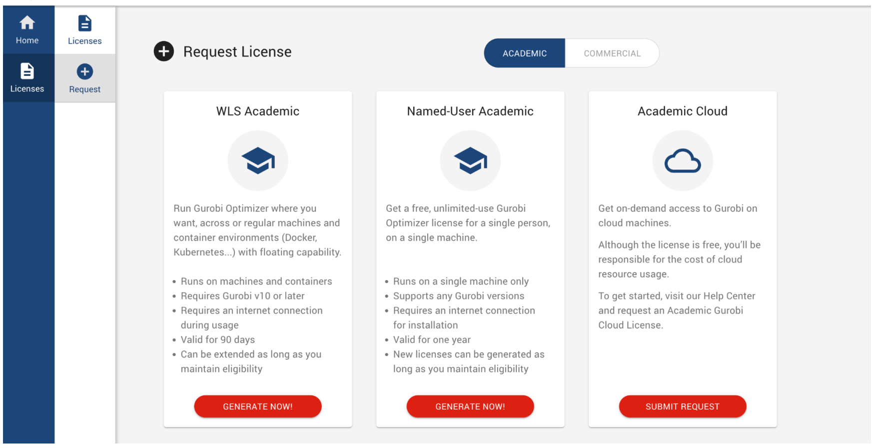
ii) Or copy the lines in the .bashrc file (or .zshrc if using zshell), and save the file

```
export GUROBI_HOME="$HOME/opt/gurobi950/linux64"
export GRB_LICENSE_FILE="$HOME/gurobi.lic"
export PATH="${PATH}:${GUROBI_HOME}/bin"
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/$HOME/usr/local/lib:/usr/local/lib
```

2. Getting the free academic license

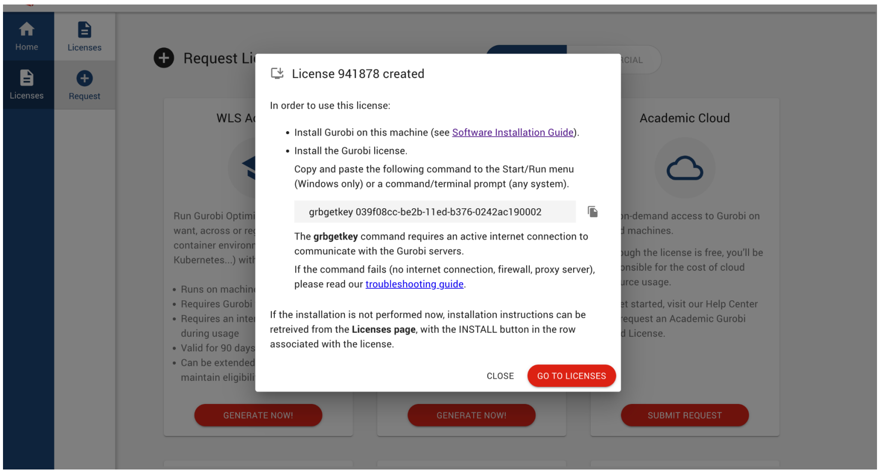
To run GUROBI one also needs to get a free academic license.
https://www.gurobi.com/documentation/9.5/quickstart_linux/retrieving_a_free_academic.html#subsection:academiclicense

a) Register using any academic email ID on the GUROBI website. b) Generate the license on https://portal.gurobi.com/iam/licenses/request/



Choose **Named-user Academic**

c) Use the command in the command prompt to generate the licesne.



(If not automatically done, place the license in one of the following locations "/opt/gurobi/gurobi.lic" or "$HOME/gurobi.lic")

Step 2: Installing Python dependencies in a virtual environment

First, make sure you have venv (https://docs.python.org/3/library/venv.html).
If venv is not already installed, install it with the following command (Use appropriate python version)

sudo apt-get install python3.8-venv

(One can also use other environments such as conda, however we have not tested the experiments on other Python environments)

To create the virtual environment,

python3 -m venv env

Then to enter the virtual environment, run

source env/bin/activate

Install all packages including the compiler with

pip install -r requirements.txt

Step 3: Running experiments

Caveats:

1. The speedup results obtained in the experiment can vary depending on the machine

2. Our experiments run our tool IVAN and the baseline on a fixed number of randomly chosen inputs from the dataset. We report the average speedup on each verification instance. The speedup results in the paper are for count=100. One can change the count=20 to a smaller value for faster run of all experiments. However, the average speedup result may vary depending on this value.

3. Speedups are also dependent on the timeout used for verification. To accurate reproduce the results from the paper, we advise not changing those timeout values otherwise the observed speedups can be less or more than the ones reported in the paper.

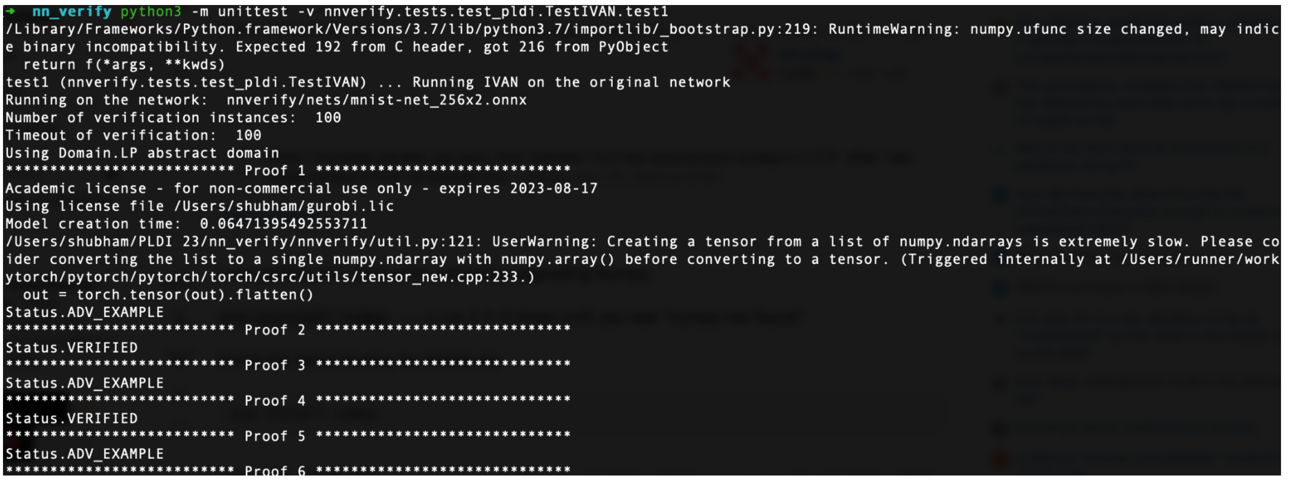
Instructions for running experiments:

- A **single experiment** runs IVAN and the baseline for verifying a set of properties on a fixed network and fixed type of modification.

  One can run a single experiment from the test using the following command. This will take about 1 to 2 hours.

  python3 -m unittest -v nnverify.tests.test_pldi.TestIVAN.test1

  Running the experiment will result in following console output



```
➜  nn_verify python3 -m unittest -v nnverify.tests.test_pldi.TestIVAN.test1
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/importlib/_bootstrap.py:219: RuntimeWarning: numpy.ufunc size changed, may indic
e binary incompatibility. Expected 192 from C header, got 216 from PyObject
  return f(*args, **kwds)
test1 (nnverify.tests.test_pldi.TestIVAN) ... Running IVAN on the original network
Running on the network:  nnverify/nets/mnist-net_256x2.onnx
Number of verification instances:  100
Timeout of verification:  100
Using Domain.LP abstract domain
************************** Proof 1 ****************************
Academic license - for non-commercial use only - expires 2023-08-17
Using license file /Users/shubham/gurobi.lic
Model creation time:  0.06471395492553711
/Users/shubham/PLDI 23/nn_verify/nnverify/util.py:121: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slow. Please co
ider converting the list to a single numpy.ndarray with numpy.array() before converting to a tensor. (Triggered internally at /Users/runner/work
ytorch/pytorch/pytorch/torch/csrc/utils/tensor_new.cpp:233.)
  out = torch.tensor(out).flatten()
Status.ADV_EXAMPLE
************************** Proof 2 ****************************
Status.VERIFIED
************************** Proof 3 ****************************
Status.ADV_EXAMPLE
************************** Proof 4 ****************************
Status.VERIFIED
************************** Proof 5 ****************************
Status.ADV_EXAMPLE
************************** Proof 6 ****************************
```

The following part of the console output present the verifier name, network name, number of inputs in the experiment and the timeout for the verification.

test1 (nnverify.tests.test_pldi.TestIVAN) ... Running IVAN on the original network

Running on the network: nnverify/nets/mnist-net_256x2.onnx
Number of verification instances: 100
Timeout of verification: 100

There are 4 possible outcomes of verification that are also printed on the console

VERIFIED - The robustness property is verified

ADV_EXAMPLE - The verifier found a adversarial example for the property

MISS_CLASSIFIED - DNN missclassified the input. This is a trivial counter-example for the property

UNKNOWN - The verifier timed out

- **All experiments** in the paper consider multiple combinations of networks and network perturbations for evaluating IVAN's effectiveness in verification compared to the baseline. The goal of experiments is to compute IVAN's speedup over the baselines.

  All experiment unit tests are located in *nnverify/test_pldi.py.* All tests can be run using the following command.
  python3 -m unittest -v nnverify.tests.test_pldi.TestIVAN

  The total running time for these tests is about 20 hours. Table 2, Figures 5, and 6 are the results of these experiments.

- **How to see the speedup results**
  The results are printed at stdout. But it is easier to check them in the file *results/proof_transfer.csv* at the end of the execution. In The csv file each experiment result is summarized in 3 lines, including speedup and extra properties verified.

```
Proof Transfer Result at,2023-03-01 14:20:44
nnverify/nets/mnist-net_256x2.onnx,Dataset.MNIST,QuantizationType.INT16,count:,20
prev branches: ,3.1999999999999998,approx branches:,1.7000000000000004
prev time: ,22.0928897857666,approx time:,11.750927448272705,speedup:,1.8800975397915578,extra completed:,0
```

  This includes information about
  a) time of the experiment b) network name c) network update type d) Number of verification instances e) time taken by IVAN and baseline f) proof tree size by IVAN and the baseline g) extra properties verified by IVAN compared to the baseline

Results with following details are also pickled in *results/pickle/* directory. This includes for i) time taken ii) verification output iii) proof tree size

- The **ablation study** experiments (Table 2) from the paper are also included in the same file. One can run those experiments in the following cases:

  Reuse →
  python3 -m unittest -v nnverify.tests.test_pldi.TestReuse
  Reorder →
  python3 -m unittest -v nnverify.tests.test_pldi.TestReorder

Similar to the previous case, the runtime is roughly 20 hours and can be made smaller by decreasing the count of verification instances.

- **Hyperparameter sensitivity** experiments (Figure 8) can be performed using
  python3 -m unittest -v nnverify.tests.test_pldi.TestSensitivity

****
Adding New Experiments
****

Similar to existing experiments one can easily add new experiments using a unit test. One can add this test in existing test file *nnverify/test_pldi.py* or can create a new test file.

More information about the adding unittests in python is available here https://docs.python.org/3/library/unittest.html.

A test function looks like following

```
def test_new(self):
    args = pt.TransferArgs(
            net='mnist_0.1.onnx',
            domain=Domain.LP,
            split=Split.RELU_ESIP_SCORE,
            approx=ap.Quantize(ap.QuantizationType.INT8)
            dataset=Dataset.MNIST,
            eps=0.02,
            count=100,
            pt_method=IVAN(0.003, 0.003),
            timeout=100)
    pt.proof_transfer(args)
```

Here,

net: location of the onnx or torch network. The networks should be placed in *nnverify/nets* directory.

domain: The abstract domain used by the verifier. The paper experiments are with LP and DeepZ domains. We have recently added more domains.

split: The branching heuristic used

approx: The modification performed on the network.

dataset: Includes MNIST, CIFAR10, and ACAS-XU

eps: Robustness radius

count: Number of verification properties

timeout: timeout for each verification instance

pt_method: This is to choose the exact incremental verification technique used in the paper. *IVAN(alpha, beta)* combines all the main techniques. *REORDERING(alpha, beta)* uses just the reordering technique, whereas *REUSE* uses just the reuse technique. The latter 2 were used for the ablation study. *alpha* and *beta* are hyperparameters that can be tuned for better results.