Overview of the Repeatability Package for “Verification of Neural Network Controllers Using Taylor Model Preconditioning”

Radoslav Ivanov, Taylor Carpenter, James Weimer,
Rajeev Alur, George Pappas, Insup Lee

1 Introduction

This document provides an overview of the repeatability package for the paper “Verification of Neural Network Controllers Using Taylor Model Preconditioning”, which has been accepted for publication at CAV’21. The paper presents a verification tool, called TMP, for verifying safety properties of autonomous systems with neural network controllers.

In summary, the package contains the following components:

- TMP: this is the name of our proposed tool;
- Verisig, ReachNN*, NNV: these are three tools we are using in the comparison evaluation;
- 10 verification benchmarks, including dynamics models and neural network controllers;
- scripts to reproduce the computational components in the paper.

2 Installation

Most of the repeatability package is contained in a Docker image. However, Matlab is required to run NNV as well as to plot some of the results. Thus, the Matlab files are included separately from the Docker image. We used Matlab R2018a in our experiments, though other versions should work as well.

The repeatability package can be found at https://github.com/rivapp/CAV21_repeatability_package.git. Note that this repository does not contain the built Docker image, but rather the Dockerfile that can be used to build the image.

A full pre-built Docker image ($> 3Gb$) can be downloaded from: https://drive.google.com/file/d/1cwXaEFJBj-3f5UQycneb48BsJcd5uXoK/view?usp=sharing.

If you would like to download the image from the command line (e.g., from a server), you can use the following command (install `gdown` using `pip install gdown`):

```
gdown https://drive.google.com/uc?id=1cwXaEFJBj-3
    ↪ f5UQycneb48BsJcd5uXoK
```

2.1 Building the Docker image

The Docker image can be built from the Dockerfile by calling (from the directory that contains the Dockerfile):

```
$ docker build -t cav21re .
```

The above command takes several minutes in order to install all the libraries and tools. Finally, it will create a Docker image named `cav21re`.

2.2 Loading the Docker image

If you elect to download the pre-built docker image, you can load it using the following command:

```
$ docker load -i cav21re.tar.gz
```

2.3 Creating the Docker container

Once you have loaded the Docker image, the container is created as follows:

```
$ docker run -it --name cav21container cav21re
```

Note that the ReachNN* tool is optimized to use an NVIDIA GPU. If your machine has an NVIDIA GPU and *nvidia-docker2* installed, you can create the container as follows:

```
$ docker run -it --name cav21container --gpus all cav21re
```

Finally, if you exit the container and would like to resume using it, you can do so as follows:

```
$ docker start -i cav21container
```

2.4 Installing NNV

Since NNV is written in Matlab, it cannot be included in the Docker image due to Matlab licensing issues. Thus, we include it separately in the `nnv` folder. Note that the code was downloaded (retrieved before the CAV'21 submission deadline) from https://github.com/verivital/nnv.git. Also note that we have only included the main engine code from the repository and have excluded other examples in the interest of space.

To install NNV, navigate to the `nnv` folder and run the `install.m` script. This will download various Matlab packages (such as the CORA reachability tool) and will add NNV to the Matlab path. Note that NNV needs several Matlab toolboxes, notably the Deep Learning toolbox.

3 TMP Overview

TMP is a tool for verifying safety properties of hybrid systems with neural network controllers. It works by approximating the neural network with a Taylor model (a polynomial with worst-case error bounds) and then using an existing reachability tool, Flow*, to propagate the Taylor model through the plant dynamics.

Note that we are the developers of both TMP and Verisig, hence the two tools have the same input-output structure. For reference, we provide the Verisig user manual in the `UserManual.pdf` file, although that is not needed to reproduce the results in this package. All TMP additions will be incorporated in the main Verisig repository (https://github.com/Verisig/verisig.git) should the paper get accepted.

TMP and Verisig both use the Flow* tool for reachability analysis. While Verisig essentially calls existing Flow* functionality, TMP contains a complete reimplementation of all relevant functions for the case of neural networks, including the newly added functionality of Taylor model preconditioning and shrink wrapping. For reference, all files in the `tmp` folder beginning with 'NN' are new.

TMP can be run in 2 ways:

1. GUI-based mode. In this mode, we use the modeling tool SpaceEx to create the hybrid system (in XML format). Then, we run a Verisig translator to convert the XML file to a Flow* model. Finally, Flow* is called. Please refer to the Verisig user manual for details.

2. Headless mode. In this mode, we directly call Flow*, assuming that the Flow* models have been already created.

4 Reproducing Table 2

Table 2 is the first computational artifact in the paper. It compares the verification results for the 4 considered tools in terms of the ability to verify a given property as well as the verification time. All properties are reachability properties: the system begins in some set of initial states, and the problem is to verify whether a certain goal state is reached. Please consult the paper for the specific numbers for each benchmark.

There are a few aspects to keep in mind:

1. For each tool we provide the verification time only if the tool terminates with a conclusive result (i.e., the output is either Safe or Unsafe). If a tool cannot verify a given property (due to too much approximation error),

it is marked as Unknown. If a tool crashed on a given benchmark, it is marked as DNF.

2. All properties are Safe (i.e., the system reaches the specified goal), except for benchmark B_5 with a sigmoid-based controller, where the correct result is Unsafe.

Important: Since some of the benchmarks require significant computation, we recommend starting with the benchmarks plotted in Figures 3-5, which take at most an hour each.

4.1 Reproducing the TMP (1 core) results

Since running a GUI from the docker is challenging, we run TMP in headless mode. All Flow* models, along with the corresponding neural network controllers (in YAML format), are provided in the `/home/verisig_models` folder in the Docker container. For reference, we also provide the XML files if one would like to inspect them locally.

TMP is installed and pre-compiled in the `/home/tmp` folder in the Docker container. In headless mode, TMP is called in the following way (from the `home` folder):

```
$ ./tmp/flowstar -p <nn_controller_yml> < <flowstarmodel>
```

Note that the -p option specifies that the tool should output a Matlab plot of reachable sets (to be used to reproduce Figures 3-5).

For completeness, we provide below the full list of commands to run in order to reproduce each entry in Table 1. Note that these benchmarks can typically be run on a standard laptop (with at least 8Gb of RAM). Please consult Table 2 in the paper to get a rough idea of what verification times to expect for the various benchmarks (for some reason, the ACC benchmark takes about an hour in the docker, as opposed to 35 minutes in a normal setting). Please run these from the `home` folder.

```
$ ./tmp/flowstar -p verisig_models/ex1_tanh/tanh20x20.yml <
    ↪ verisig_models/ex1_tanh/ex1_tanh_tmp.model
$ ./tmp/flowstar -p verisig_models/ex1_sig/sig20x20.yml <
    ↪ verisig_models/ex1_sig/ex1_sig_tmp.model
$ ./tmp/flowstar -p verisig_models/ex2_tanh/tanh20x20.yml <
    ↪ verisig_models/ex2_tanh/ex2_tanh_tmp.model
$ ./tmp/flowstar -p verisig_models/ex2_sig/sig20x20.yml <
    ↪ verisig_models/ex2_sig/ex2_sig_tmp.model
$ ./tmp/flowstar -p verisig_models/ex3_tanh/tanh20x20.yml <
    ↪ verisig_models/ex3_tanh/ex3_tanh_tmp.model
$ ./tmp/flowstar -p verisig_models/ex3_sig/sig20x20.yml <
    ↪ verisig_models/ex3_sig/ex3_sig_tmp.model
$ ./tmp/flowstar -p verisig_models/ex4_tanh/tanh20x20.yml <
    ↪ verisig_models/ex4_tanh/ex4_tanh_tmp.model
```

```
$ ./tmp/flowstar -p verisig_models/ex4_sig/sig20x20.yml <
    ↪ verisig_models/ex4_sig/ex4_sig_tmp.model
$ ./tmp/flowstar -p verisig_models/ex5_tanh/tanh100x100x100.yml <
    ↪  verisig_models/ex5_tanh/ex5_tanh_tmp.model
$ ./tmp/flowstar -p verisig_models/ex5_sig/sig100x100x100.yml <
    ↪ verisig_models/ex5_sig/ex5_sig_tmp.model
$ ./tmp/flowstar -p verisig_models/tora_tanh/tanh20x20x20.yml <
    ↪ verisig_models/tora_tanh/tora_tanh_tmp.model
$ ./tmp/flowstar -p verisig_models/tora_sig/sig20x20x20.yml <
    ↪ verisig_models/tora_sig/tora_sig_tmp.model
$ ./tmp/flowstar -p verisig_models/acc/tanh20x20x20.yml <
    ↪ verisig_models/acc/ACC_tanh_tmp.model
$ ./tmp/flowstar -p verisig_models/mountain_car_small/sig16x16.
    ↪ yml < verisig_models/mountain_car_small/MC_small_sig_tmp.
    ↪ model
$ ./tmp/flowstar -p verisig_models/mountain_car_large/sig200x200.
    ↪ yml < verisig_models/mountain_car_large/MC_large_sig_tmp.
    ↪ model
$ ./tmp/flowstar -p verisig_models/quadrotor_mpc/tanh20x20.yml <
    ↪ verisig_models/quadrotor_mpc/quadrotor_MPC_tanh_tmp.model
$ ./tmp/flowstar -p ./verisig_models/f1tenth/tanh64x64.yml < ./
    ↪ verisig_models/f1tenth/f1tenth_tanh_tmp.model
```

4.1.1 Inspecting TMP's output

Once the tool completes a given benchmark, it will 1) print to the console the verification result (SAFE, UNSAFE or UNKNOWN) as well as the verification time and 2) generate a Matlab file (in the `outputs` folder) that will be used to plot reachable sets.

4.2 Reproducing the TMP (40 cores) results

If one has access to a multi-core machine, one can use the parallelized nature of TMP. Specifying the number of cores when running TMP is done using the -t option (the default is 1). Specifically, the -t options specifies the number of threads that will be spawned during the neural network verification part of the computation. For example, calling TMP with 40 cores on benchmark B_1 with a tanh-based controller can be done as follows (from the `/home` directory):

```
$ time ./tmp/flowstar -t 40 -p verisig_models/ex1_tanh/tanh20x20.
    ↪ yml < verisig_models/ex1_tanh/ex1_tanh_tmp.model
```

For proper timing of the verification computation, we use the *time* command; the verification cost output by TMP at the end of the computation is the total computation time across all cores, which is not reported in Table 2.

4.3 Reproducing the Verisig results

Verisig is installed and pre-compiled in the `/home/verisig` folder. The tool's usage is the same as TMP's. The source code is available at https://github.com/Verisig/verisig.git. Note that we use separate Flow* model files for TMP and Verisig, the only difference being the name of the output plotting files (used to reproduce Figures 3-5). For completeness, we provide below all commands used to reproduce the Verisig results in Table 2 (run from the `/home` folder).

Warning: Verisig takes more than 10 hours on the F1/10 benchmark and requires more than 10Gb of RAM. Also note that Verisig crashes on benchmark B_1 with a tanh controller, as indicated in Table 2 in the paper.

```
$ ./verisig/flowstar/flowstar -p verisig_models/ex1_tanh/
    ↪ tanh20x20.yml < verisig_models/ex1_tanh/ex1_tanh_verisig.
    ↪ model
$ ./verisig/flowstar/flowstar -p verisig_models/ex1_sig/sig20x20.
    ↪ yml < verisig_models/ex1_sig/ex1_sig_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/ex2_tanh/
    ↪ tanh20x20.yml < verisig_models/ex2_tanh/ex2_tanh_verisig.
    ↪ model
$ ./verisig/flowstar/flowstar -p verisig_models/ex2_sig/sig20x20.
    ↪ yml < verisig_models/ex2_sig/ex2_sig_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/ex3_tanh/
    ↪ tanh20x20.yml < verisig_models/ex3_tanh/ex3_tanh_verisig.
    ↪ model
$ ./verisig/flowstar/flowstar -p verisig_models/ex3_sig/sig20x20.
    ↪ yml < verisig_models/ex3_sig/ex3_sig_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/ex4_tanh/
    ↪ tanh20x20.yml < verisig_models/ex4_tanh/ex4_tanh_verisig.
    ↪ model
$ ./verisig/flowstar/flowstar -p verisig_models/ex4_sig/sig20x20.
    ↪ yml < verisig_models/ex4_sig/ex4_sig_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/ex5_tanh/
    ↪ tanh100x100x100.yml < verisig_models/ex5_tanh/
    ↪ ex5_tanh_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/ex5_sig/
    ↪ sig100x100x100.yml < verisig_models/ex5_sig/
    ↪ ex5_sig_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/tora_tanh/
    ↪ tanh20x20x20.yml < verisig_models/tora_tanh/
    ↪ tora_tanh_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/tora_sig/
    ↪ sig20x20x20.yml < verisig_models/tora_sig/tora_sig_verisig
    ↪ .model
$ ./verisig/flowstar/flowstar -p verisig_models/acc/tanh20x20x20.
    ↪ yml < verisig_models/acc/ACC_tanh_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/
```

```
    ↪ mountain_car_small/sig16x16.yml < verisig_models/
    ↪ mountain_car_small/MC_small_sig_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/
    ↪ mountain_car_large/sig200x200.yml < verisig_models/
    ↪ mountain_car_large/MC_large_sig_verisig.model
$ ./verisig/flowstar/flowstar -p verisig_models/quadrotor_mpc/
    ↪ tanh20x20.yml < verisig_models/quadrotor_mpc/
    ↪ quadrotor_MPC_tanh_verisig.model
$ ./verisig/flowstar/flowstar -p ./verisig_models/f1tenth/
    ↪ tanh64x64.yml < ./verisig_models/f1tenth/
    ↪ f1tenth_tanh_verisig.model
```

4.3.1 Inspecting Verisig's output

Verisig's output is the same as TMP's. Upon termination, it will 1) print to the console the verification result (SAFE, UNSAFE or UNKNOWN) as well as the verification time and 2) generate a Matlab file (in the `outputs` folder) that will be used to plot reachable sets.

4.4 Reproducing the ReachNN* results

As noted earlier, ReachNN* is optimized for GPU performance. If you run it without a GPU, it will take considerably longer to run, especially on the Tora benchmarks. Note that the Tora benchmarks also require more than 10Gb of RAM and may crash otherwise.

The ReachNN* tool is installed and pre-compiled in the `/home/reachNNStar` folder. We downloaded the code from https://github.com/JmfanBU/ReachNNStar.git. A few notes:

- we modified the tool to output Matlab plot files, as opposed to the Gnuplot files that were used in the original repository;
- ReachNN* cannot be used on the MC, QMPC and F1/10 benchmarks since they have hybrid plant dynamics;
- after some discussion with the ReachNN* authors, it was determined that the tool cannot scale to the ACC benchmark.

For completeness, we provide below all commands to reproduce all ReachNN* results in Table 2 (run from the `/home/reachNNStar/ReachNN` folder):

```
$ ./run_ex1_tanh.sh
$ ./run_ex1_sig.sh
$ ./run_ex2_tanh.sh
$ ./run_ex2_sig.sh
$ ./run_ex3_tanh.sh
$ ./run_ex3_sig.sh
$ ./run_ex4_tanh.sh
$ ./run_ex4_sig.sh
$ ./run_ex5_tanh.sh
$ ./run_ex5_sig.sh
$ ./run_tora_tanh.sh
$ ./run_tora_sig.sh
```

4.4.1 Inspecting ReachNN* outputs

Once the tool terminates, it will create 1) a *.txt* file containing the verification outcome as well as verification time and 2) a Matlab file to be used for plotting reachable sets. Both of these are stored in the `/home/reachNNStar/ReachNN/outputs` folder. You can inspect the *.txt* files to view the result for each benchmark. For example, to view the outcome of running B_1 with a tanh-based controller, you can view the output as follows:

```
$ emacs outputs/nn_1_tanh.txt
```

Finally, press CTRL-X CTRL-C to exit Emacs.

4.5 Reproducing the NNV results

As noted above, NNV uses Matlab, so it cannot be run from inside the Docker container due to Matlab licensing issues.

To run NNV, first make sure it is properly installed and added to the Matlab path as explained in the Installation section. Then navigate to the `nnv/examples/CAV Comparisons` folder, which contains a list of all the benchmarks. For each benchmark, it is sufficient to run the *run_system.m* script from within that benchmark's folder. For example, to run the ACC benchmark, navigate to the `nnv/examples/CAV Comparisons/acc` folder and run *run_system.m*. A few notes:

- note that NNV crashes on benchmarks B_1 and B_4 due to bugs;
- each *run_system.m* file is separated into cells. For some benchmarks, it might be better to run the individual cells separately, since they may require a lot of RAM;
- note that NNV cannot handle hybrid plant dynamics, hence it cannot be run on the MC, QMPC and F1/10 benchmarks.

4.5.1 Inspecting NNV's outputs

At the end of the computation, NNV will 1) print the verification result and the verification time to the console and 2) save the reach sets to a 'mat' file. This file will be used to plot reachable sets (Figures 3-5), as explained next.

5 Reproducing Figures 3-5

Figures 3-5 provide a comparison between the various tools in terms of reachable sets. Producing the plots is a bit cumbersome since the different tools have different output formats.

First, we need to pre-process each tool's output files in order to prepare them for plotting. In particular, the *clean_up.py* script in the `cleanup_scripts` folder is used to modify the various Matlab output files by fixing the colors and removing some unnecessary lines. If you followed the instructions in the previous section, then all output files should be in the `/home/outputs` folder (in the case of Verisig and TMP) and in the `/home/reachNNStar/ReachNN/outputs` folder (in the case of ReachNN*). To clean up the output files, run the following two commands (from the `cleanup_scripts` folder), which will store the processed Matlab files in a new `/home/outputs_clean` folder.

```
$ python3 clean_up.py ../outputs ../outputs_clean
$ python3 clean_up.py ../reachNNStar/ReachNN/outputs ../
    ↪ outputs_clean
```

Next, we need to transfer the cleaned up files from the Docker to the host machine, in order to plot them in Matlab. The most convenient way to transfer files is to use the `docker cp` command, which takes the following form:

```
$ docker cp <containerId>:/file/path/within/container /host/path/
    ↪ target
```

To get the container id, one can run `docker ps`, which will show the id's of all running containers.

Figures 3-5 consist of 9 plots altogether. Each plot is generated in a separate folder in the `plots` folder, e.g., Figure 3a is plotted in the `plots/Fig3a` folder. In addition to the two reachable set graphs, each plot also shows example simulated trajectories for that system. To reproduce each plot, one needs to do the following from each of the 9 folders:

1. add the `plots/altmany-export_fig` folder to the Matlab path. This package is used to save the final plots to pdf files;
2. generate simulated trajectories using the *sim_trajectories.m* in each plot folder; the trajectories will be stored in a file *trajectories.mat*;
3. transfer the cleaned up output files to the plot folder;
4. run the *plot_reach_sets.m* script to plot the figures and save them in a pdf file.

Note that we provide back-up cleaned up plot files in each folder as examples.

To plot each subfigure in Figure 3, please run the following commands (assuming the docker container is running on the same machine). Figure 3a (folder `plots/Fig3a`):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/acc_tanh_tmp.m .
$ docker cp <containerId>:/home/outputs_clean/acc_tanh_verisig.m
    ↪ .
//run plot_reach_sets.m
```

Figure 3b (folder `plots/Fig3b`):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/ex5_sig_tmp.m .
$ docker cp <containerId>:/home/outputs_clean/ex5_sig_verisig.m .
//run plot_reach_sets.m
```

Figure 3c (folder `plots/Fig3c` folders, respectively):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/MC_large_sig_tmp.m
    ↪ .
$ docker cp <containerId>:/home/outputs_clean/
    ↪ MC_large_sig_verisig.m .
//run plot_reach_sets.m
```

To plot each subfigure in Figure 4, please run the following commands (assuming the docker container is running on the same machine).
Figure 4a (folder `plots/Fig4a`):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/ex1_sig_tmp.m .
$ docker cp <containerId>:/home/outputs_clean/nn_1_sigmoid.m .
//run plot_reach_sets.m
```

Figure 4b (folder `plots/Fig4b`):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/ex5_tanh_tmp.m .
$ docker cp <containerId>:/home/outputs_clean/nn_5_tanh.m .
//run plot_reach_sets.m
```

Figure 4c (folder `plots/Fig4c`):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/tora_sig_tmp.m .
$ docker cp <containerId>:/home/outputs_clean/nn_tora_sigmoid.m .
//run plot_reach_sets.m
```

To plot each subfigure in Figure 4, please run the following commands (assuming the docker container is running on the same machine and `nnv` is in the Matlab path).
Figure 5a (folder `plots/Fig5a`):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/acc_tanh_tmp.m .
$ cp ../nnv/examples/CAV Comparisons/acc/nnv_trajectories.mat .
//run plot_reach_sets.m
```

Figure 5b (folder `plots/Fig5b`):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/ex2_sig_tmp.m .
$ cp ../nnv/examples/CAV Comparisons/ex2_sig/nnv_trajectories.mat
    ↪  .
//run plot_reach_sets.m
```

Figure 5c (folder `plots/Fig5c`):

```
//run sim_trajectories.m
$ docker cp <containerId>:/home/outputs_clean/tora_tanh_tmp.m .
$ cp ../nnv/examples/CAV Comparisons/tora_tanh/nnv_trajectories.
    ↪ mat .
//run plot_reach_sets.m
```

Note that NNV does not directly output Matlab plot files, so instead we load the object containing all the NNV reach sets (from *nnv_trajectories.mat*) and then we use the NNV plotting functionality.