# Artifact for Interactive Bit Vector Reasoning using Verified Bitblasting

## .1 Artifact check-list (meta-information)

## A Introduction

This artifact contains the infrastructure and tooling necessary to display the performance of the verified bitblaster introduced by the paper for all the benchmarks presented.

- **Program:** The code repository for our framework along with the test suite. Note that this is already setup in the docker image.
- **Compilation:** The Lean4 toolchain, downloaded via `elan`. Note that this is already setup in the docker image.
- **Run-time environment:** Any operating system that supports Docker.
- **Hardware:** Any x86-64 machine.
- **Output:** Key theorems of the paper will be built and shown to have no unsound axioms.
- **How much disk space required (approximately)?:** 30GB
- **How much time is needed to prepare workflow (approximately)?:** 1hr
- **How much time is needed to complete experiments (approximately)?:** 5hr
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** MIT
- **Archived (provide DOI)?:** 10.5281/zenodo.15755236

### A.1 Performance

We test the performance of the verified bitblaster `bv_decide` against three benchmarks:

- `InstCombine` benchmark, extracted from LLVM's peephole verifier
- `HackersDelight` benchmark, containing bit-vector theorems extracted from the first two chapters of Hackers' Delight
- `SMT-LIB` benchmark, containing the problems from SMT-LIB's 2024 Competition

The artifact contains the benchmarks, the scripts to evaulate `bv_decide`'s performance, and the scripts to reproduce the plots in the paper.

## B Hardware Dependencies

Podman or Docker are necessary to run our artifact (we tested it with the former). The container image has all dependencies needed to compile our framework with Lean4. TODO: add info

## C Getting Started

Access the docker image from 10.5281/zenodo.15755236:

```
$ docker load -i oopsla25-bv-decide.tar
$ docker run -it oopsla25-bv-decide
$ cd /code/lean-mlir && lake clean && lake exe cache get && lake build
# Run experiments InstCombine and HackersDelight
# Collect the output and write all the plots
$ /code/lean-mlir/artifacts/oopsla25-bv-decide/run.sh
```

Author's Contact Information:

## D Experiments Reproduction

Three main scripts are involved in the reproduction of our results and plots:

- `compare.py benchmark` runs our bitblaster as well as the solver we compare against for `benchmark`. The results obtained from this run are saved in `bv-evaluation/results`.
- `collect.py benchmark` collects and analyzes the results obtain for `benchmark` and stores everything in `bv-evaluation/raw-data`.
- `plot.py benchmark` plots the results obtained from `benchmark`'s run, including the plots presented in the paper.

### D.1 `InstCombine` - Reproduce Fig. 9

As an example, to reproduce Fig. 9 using 8 threads for the experimental run and 5 repetitions for stability, the sequence of commands to run are:

```
$ docker load -i oopsla25-bv-decide.tar
$ docker run -it oopsla25-bv-decide
$ cd /code/lean-mlir && lake clean && lake exe cache get && lake build
# Run experiments for InstCombine
$ /code/lean-mlir/bv-evaluation/compare.py instcombine -j8 -r5
# Collect InstCombine data
$ /code/lean-mlir/bv-evaluation/collect.py instcombine
# Plot InstCombine data
$ /code/lean-mlir/bv-evaluation/plot.py instcombine
```

The expected plot is in `plots/bv_decide_stacked_perc_instCombine.pdf`