PSOUFFLE Artifact

Artifact for the FMCAD 2026 paper *PSOUFFLE: Towards Scalable Probabilistic Logic Inference for Program Analysis*.

The Zenodo Docker image is the runtime environment for artifact evaluation. It contains the benchmark workspace at `/workspace`, ProbLog installed as a Python package, and PSouffle installed as `/usr/local/bin/souffle`. `FMCAD.py` is the single entry point for the three research questions. The appendix at the end of this document explains the PSouffle pipeline and the output TSV formats in more detail.

For a deeper understanding of PSouffle behavior, evaluators can also read `psouffle.zip`.

Image Contents

The Docker image provides:

- Python 3.10 and the Python packages used by the runner.
- ProbLog (profiling branch), CUDD, and SDD++ for probabilistic inference.
- PSouffle.
- VProbLog.
- Three benchmark trees pre-staged under `/workspace`:
  - `side_channel/full/P{1,3..20}/` (sc, 19 cases)
  - `taint/<case>/` (taint, 15 Android cases)
  - `symbolization/<case>/` (dis, 16 DDisasm cases)

Build

```
unzip fmcad26-artifact.zip
cd fmcad26-artifact
docker build -t fmcad26-artifact .
```

Build takes about half an hour. Run all docker commands below from this directory.

Fast Verification From Pre-recorded Logs

The zip already includes `artifact/runs/rq{1,2,3}/RQ*result.tsv`. To rebuild the TSVs in place and confirm the collector matches the paper:

```
docker run --rm -it \
  -v "$PWD/artifact:/workspace/artifact" \
  fmcad26-artifact \
  sh -c 'python3 FMCAD.py -RQ1 --rq1-collect && \
          python3 FMCAD.py -RQ2 --rq2-collect && \
          python3 FMCAD.py -RQ3 --rq3-collect'
```

Smoke Check

Run the smoke check next. It runs PSouffle + ProbLog on side-channel `P1` and `P3` with one run each, and verifies that the runner, bundled cases, and binaries are wired correctly. Estimated runtime: 2–5 minutes.

```
docker run --rm -it \
  -v "$PWD/artifact:/workspace/artifact" \
  fmcad26-artifact \
  python3 FMCAD.py --smoke
```

A successful run leaves:

```
artifact/smoke/P1/...
artifact/smoke/P3/...
artifact/smoke/FMCADresult.tsv
```

`FMCADresult.tsv` lists per-case PSouffle and ProbLog wall-clock times. If both P1 and P3 appear with non-error status, the artifact is healthy.

Run Representative

The representative workload runs `-RQ2` on five cases per benchmark (sc / taint / dis) with one run per (case, engine) and a 5-minute timeout per cell. It exercises all three engines (PSouffle, ProbLog, VProbLog) on the same probabilistic programs. Estimated runtime: under 1 hour (45 cells $\times$ $\leq$5 min).

```
docker run --rm -it \
  -v "$PWD/artifact:/workspace/artifact" \
  fmcad26-artifact \
  sh -c 'python3 FMCAD.py --representative && \
          python3 FMCAD.py -RQ2 --rq2-base-dir artifact/representative --rq2-collect'
```

Inspect the runtime comparison table from the host:

```
sed -n "1,20p" artifact/representative/RQ2result.tsv
```

The five sc cases are P1, P3, P4, P5, P6. The five taint cases are `and-roc`, `andors-trail`, `angulo`, `app-018`, `app-ca7`. The five dis cases are `bison`, `cluster`, `flex`, `gawk`, `gcc11`.

Selective: Reproduce a Single RQ

To rerun just one paper item, invoke its RQ flag. Output overwrites `artifact/runs/rq{1,2,3}/` as before; `--resume` skips finished cells.

```
docker run --rm -it -v "$PWD/artifact:/workspace/artifact" fmcad26-artifact \
  python3 FMCAD.py -RQ1     # Tables I & II  (~2 h)
docker run --rm -it -v "$PWD/artifact:/workspace/artifact" fmcad26-artifact \
  python3 FMCAD.py -RQ3     # Figure 2        (~6 h)
docker run --rm -it -v "$PWD/artifact:/workspace/artifact" fmcad26-artifact \
  python3 FMCAD.py -RQ2     # Table III       (~24 h)
```

Selective: Run Complete

The full workload reproduces every paper table and figure — RQ1 on every case (graph metrics), RQ3 on every case (rewrite on/off runtime), then RQ2 across all engines. **Plan for >24 hours of wall time** on an 8-core 32 G machine; RQ2 alone can take 12–24 h because ProbLog/VProbLog hit the per-cell timeout on hard cases.

```
docker run --rm -it \
  -v "$PWD/artifact:/workspace/artifact" \
  fmcad26-artifact \
  sh -c 'python3 FMCAD.py -RQ1 && \
          python3 FMCAD.py -RQ3 && \
          python3 FMCAD.py -RQ2'
```

The complete run overwrites `artifact/runs/rq{1,2,3}/`. To pick up an interrupted run, repeat the same command with `--resume` appended to each invocation.

Read the Results

Each TSV under `artifact/` corresponds to one paper item. The `Status` column is `OK`, `TIMEOUT`, or `OOM`; runtime columns are wall-clock seconds.

File	Paper item	How to read it
`runs/rq1/RQ1result.tsv`	Table I	Per-case derivation-graph node and edge counts before and after the PSouffle rewrite.
`runs/rq1/RQ1summary.tsv`	Table II	Per-category aggregates (avg / max / median) of the RQ1 reduction rates.
`runs/rq2/RQ2result.tsv`	Table III	Per-(benchmark, case, engine) wall-clock runtime for PSouffle, ProbLog, and VProbLog on the same probabilistic program.
`runs/rq3/RQ3result.tsv`	Figure 2	Per-case PSouffle runtime with rewrite off (`01`) vs on (`11`); the speedup column is `01 / 11`.

Appendix: How the Artifact Works

This section explains the execution model behind the commands above. It is background only; it does not add evaluation steps.

Per-case Layout

Every benchmark case ships a self-contained directory:

```
<benchmark>/<case>/
|-- compute.souffle.dl    # Datalog program, PSouffle dialect
|-- compute.problog.dl    # same program rewritten in ProbLog syntax
|-- input/                # input relations (.facts) and per-tuple probabilities (.prob)
`-- vproblog/             # bridge form (rules + edb.conf) for VProbLog
```

Engines and Pipelines

For each (case, engine) cell, `FMCAD.py` runs one of three pipelines and writes a `run.meta.json` next to the engine output:

- **PSouffle** — compiles `compute.souffle.dl` into an executable with `souffle -F input -D output compute.souffle.dl -o compute_*`, runs the executable with the requested `--rewrite` / `--det-opt` flags, and parses the structured `log.txt_*.json`. The engine reads facts/probabilities from the `input/` directory.
- **ProbLog** — invokes `problog` directly on `compute.problog.dl`. No pre-compile step.
- **VProbLog** — invokes `vlog` on the `vproblog/`. Input and output files are in the `storemat/`.

What Each RQ Measures

RQ	Engines	Inputs	Output stage
RQ1	PSouffle	`compute.souffle.dl` + `input/`	derivation-graph node / edge counts before vs after the PSouffle rewrite
RQ3	PSouffle	same as RQ1	wall-clock runtime with rewrite off (`01`) vs on (`11`)
RQ2	PSouffle / ProbLog / VProbLog	each engine's own program form	wall-clock runtime per engine, per case

RQ1 reads node/edge counts from the PSouffle log; RQ2/RQ3 read wall-clock from `run.meta.json`. The collector aggregates these into the TSVs listed under *Read the Results*.