USENIX Security '26 Artifact Appendix: Principled Design of Indexing Functions for Memory Coloring

Stephan Dübler[1], Jana Hofmann[1], Boris Köpf[2], Stavros Volos[2]

[1]Max Planck Institute for Security and Privacy (MPI-SP)
[2]Azure Research, Microsoft

A Artifact Appendix

A.1 Abstract

Our work presents algorithms for synthesizing indexing functions that support memory coloring for partitioning of microarchitectural resources. This artifact contains the code and data for our case studies (presented in Section 7 of the paper).

In the paper, we conduct three experiments: 1) an ablation study in which we synthesize functions for increasingly complicated demands 2) an experiment in which we synthesize functions for a realistic 16-core server-class CPU and 3) a security study, in which we confirm the isolation guarantees provided by our approach.

At the top level, the experimental pipeline proceeds in two stages: 1) synthesize an indexing function from a set of address traces representing accesses to the L3 cache and 2) evaluate the performance of the synthesized function. Both the collection of L3 access traces and evaluation of performance is carried out in ChampSim, a trace-based simulator of the memory hierarchy.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact is provided as a docker container. Running it does not require any particular system setup. The only risk is that a full-fetched simulation will require significant resources for several days.

A.2.2 How to access

The artifact is available on Zenodo: https://doi.org/10.5281/zenodo.18494189. The artifact contains:

- The implementation of our algorithms and various configuration files in `function_synthesis_algorithm.tar`
- Our adapted version of the ChampSim simulator in `Champsim.tar`, which also includes our benchmarks (i.e., workload traces) in `traces`.
- A folder `scripts` with various scripts for function generation and evaluation.
- All synthesized functions produced for the paper (`synthesized_functions_paper.txt` in `synthesized_functions`) and the corresponding full evaluation metrics (`Camera-Ready-Numbers.xlsx`, `Camera-Ready-Numbers.md`)
- A Docker image (Dockerfile , docker-compose.yml)
- A `README.md` file with further information.

A.2.3 Hardware dependencies

Fully reproducing the results provided in Section 7 requires significant compute as we simulate the full memory hierarchy for several millions of instructions. By default, our scripts run 8 such simulations in parallel. We recommend a system with at least 10 CPU cores, 20GiB RAM and 60GiB free disk space for the Docker container to run. If these requirements cannot be met, the number of parallel simulations can be configured (see Section A.5). On Linux, the Docker container's available RAM scales automatically with the system. On macOS, it needs to be configured, e.g., via Docker Desktop.

A.2.4 Software dependencies

The artifact is provided as a docker container and uses docker compose. It has been tested on macOS 26.5. For instructions on how to install docker and docker compose see https://docs.docker.com/compose/install/. When run outside the docker container, the artifact requires `Python 3.11` with the following packages: mpmath 1.3.0, numpy 2.3.2, openpyxl 3.1.5, psutil 7.2.1, PyYAML 6.0.2, scipy 1.16.2, sortedcontainers 2.4.0, sympy 1.14.0

A.2.5 Benchmarks

All workload traces used in the artifact can be found in `Champsim.tar/traces`. These are SPEC2017 and PARSEC ChampSim traces. Our pipeline first generates the L3 access

traces from these workloads, which are then aggregated into a probability distribution.

A.3 Set-up

A.3.1 Installation

We recommend using the Docker image for evaluation. Install Docker and Docker Compose (see above). At the top-level of the artifact run:

```
# 0) Unpack all 4 tar archives
# 1) Build the image
docker compose build
# 2) Start the container
docker compose up -d
# 3) Open a shell inside
docker compose exec artifact bash
```

To generate the necessary training data for all following scripts, either run the full-length script `scripts/generate_training_data.sh` (approx. 12-14 hours) or the shortened variant `scripts/generate_training_data.sh --reduce-sim-instr` (approx. 3-4 hours). To closely reproduce the paper's experiments, we recommend running the full-length script.

A.3.2 Basic Test

In the Docker shell, run `scripts/full_pipeline_basic_test.sh`. This briefly starts all components of our pipeline (generation of probability distribution, function synthesis, and function evaluation). It should not take longer than 1 minute.

A.4 Evaluation workflow

The goal of this artifact is to reproduce the results of **EXP1 - EXP6** (which make claims about the performance of the synthesized functions) as well as the security evaluation (**EXP7** - Section 7.4) of the paper.

A.4.1 Major Claims

The major claims are:

Claim 1 Without coloring requirements, the synthesized functions achieve a performance that is on par with the default L3 set indexing function (within 0.5% difference for hit rate, miss latency and IPC).

Claim 2 With additional coloring requirements, the hit rate gradually decreases.

Claim 3 For a 16-core server class CPU, we can synthesize an L3 function that enables simultaneously partitioning both the L3 cache and DRAM. When enforcing 16 colors, the IPC when using the synthesize functions is reduced by less than 3%.

Claim 4 The coloring scheme obtained from the synthesized functions prevents cross-color L3 cache and row buffer evictions.

A.4.2 Experiments

We propose experiments **EXP1** - **EXP7** mirroring those described in the paper. Each experiment consists of several batches, where in each batch, the algorithm synthesizes a function from randomly selected workloads and evaluates it on different randomly selected workloads.

For the human, each experiment consists of three steps: 1) run a script, 2) pull the results from the Docker container, and 3) inspect the results in `Newly-Calculated_Numbers.xlsx` and check if they match the results given in the paper for the respective experiment.

Every script invokes a chain of scripts that executes all parts of the pipeline. Further information can be found in `README.md`.

Faithfully reproducing all experiments at the scale described in the paper takes at least 9 days if the Docker container has the resources described in Section A.2.3. For each experiment, we therefore provide a full version and a shortened version. The shortened version runs the experiment with fewer simulated instructions (detailed below).

By default, the results are inside the Docker container. To inspect them from host, either first run `exit` from the same terminal and then `scripts/extract_test_results.sh`, or open a separate terminal and run `scripts/extract_test_results.sh` directly.

Runtime estimates calibrated against measured runs on a 12-core Apple M2 Pro inside Docker Desktop with 20 GiB of allocated memory with 8 jobs in parallel.

EXP1 - supporting Claim 1 *[5 human-minutes + 11-13 compute-hours]*: Run `scripts/run_full_pipeline.sh --exp1` (approx. 11-13 hours) or `scripts/run_full_pipeline.sh --exp1 --reduce-sim-instr` (approx. 3-4 hours). The metrics produced by the synthesized functions should differ less than 0.5% compared to the default L3 function.

EXP2 - supporting Claim 1 *[5 human-minutes + 11-13 compute-hours]*: Run `scripts/run_full_pipeline.sh --exp2` (approx. 11-13 hours) or `scripts/run_full_pipeline.sh --exp2 --reduce-sim-instr` (approx. 3-4 hours). As for **EXP2**, the metrics produced by the synthesized function should differ less than 0.5% compared to the default L3 function.

EXP3 - supporting Claim 2 *[5 human-minutes + 40-50 compute-hours]*: Run `scripts/run_full_pipeline.sh --exp3` (approx. 40-50 hours) or `scripts/run_full_pipeline.sh --exp3 --reduce-sim-instr` (approx. 10-12 hours). The metrics should be within the range reported in Fig. 5 of the paper.

EXP4 - supporting Claim 2 *[5 human-minutes + 25-30*

compute-hours]: Run `scripts/run_full_pipeline.sh --exp4` (approx. 25-30 hours) or `scripts/run_full_pipeline.sh --exp4 --reduce-sim-instr` (approx. 7-9 hours). The metrics should be close to the ones reported in Fig. 6 in the paper.

EXP5 - supporting Claim 2 *[5 human-minutes + 30-35 compute-hours]*: Run `scripts/run_full_pipeline.sh --exp5` (approx. 30-35 hours) or `scripts/run_full_pipeline.sh --exp5 --reduce-sim-instr` (approx. 9-11 hours). The metrics should be close to those reported in Fig. 7 in the paper.be

EXP6 - supporting Claim 3 *[5 human-minutes + 70-75 compute-hours]*: Run `scripts/run_full_pipeline.sh --exp6` (approx. 70-75 hours) or `scripts/run_full_pipeline.sh --exp6 --reduce-sim-instr` (approx. 20-25 hours). The metrics should be close to the ones reported in Fig. 8 in the paper.

EXP7 - supporting Claim 4 *[5 human-minutes + 20-25 compute-hours]*: Run `scripts/run_full_pipeline.sh --exp7` (approx. 20-25 hours) or `scripts/run_full_pipeline.sh --exp7 --reduce-sim-instr` (approx. 6-8 hours). This experiment implements a coloring scheme derived from one of the functions synthesized in **EXP6**. The script records the evictions caused by an attacker process for both L3 and DRAM row buffers with and without coloring enabled. The results support Fig. 9 in the paper.

The exact locations of the results within `Newly-Calculated-Numbers.xlsx` are shown in Table 1. The results are duplicated in the markdown file.

Exp.	Result location in **`Newly-Calculated-Numbers.xlsx`**
Exp1	`Exp1-5-Single-Core *` sheets(1,2,3) - top sub-table cols B (`standard L3`) + C (`Performance`), rows 2-33.
Exp2	`Exp1-5-Single-Core *` sheets(1,2,3) - prefetcher sub-table cols B + C (rows 38-69).
Exp3	`Exp1-5-Single-Core *` sheets(1,2,3) - top sub-table cols E-P (`4k page Ncolors`), rows 2-33.
Exp4	`Exp1-5-Single-Core *` sheets(1,2,3) - top sub-table cols Q-X (`4kpage_dram_Ncolors`), rows 2-33.
Exp5	`Exp1-5-Single-Core *` sheets(1,2,3) - slicing sub-table cols H (`standard L3`) + I-P (`4kpage_dram_slice_Ncolors`), rows 38-69.
Exp6	`Exp6-Multi-Core` sheet(4) - col B (`standard L3 2mb page`) + cols C onward (`2mbpage_dram_slice_Ncolors`), across the Hit-Rate / IPC / Latency sub-sections.
Exp7	`Exp7-Color-Isolation` sheet(5) - LLC + DRAM eviction matrices.

Table 1: Result location in the xlsx file by experiment.

Number of Simulated Instructions Here, we provide more detailed information on the number of simulated instructions in the full (default) vs the short (`--reduce-sim-instr`) version of the above experiments.

- **EXP1 - EXP5**: Full version: 50M warmup, 200M sim. (for both training and evaluation). Short version: training: 10M warmup, 50M sim.; evaluation: 10M warmup, 20M sim.
- **EXP6**: Full version: training: 50M warmup, 100M sim.; evaluation: 50M warmup, 50M sim. Short version: training: 10M warmup, 25M sim.; evaluation: 10M warmup, 10M sim.
- **EXP7**: Full version: no warmup, 50M sim. Short version: no warmup, 1M sim.

A.5 Notes on Reusability

Pipeline Each experiment script partially invokes `scripts/run_full_pipeline.sh`, which can additionally be run with an `--jobs <i>` flag to set the number of parallel simulations to `i`. When intending to run all experiments at once, this script can be run without additional flags, also compatible with `--reduce-sim-instr`. Further details on the functionality of the provided scripts can be found in `README.md`.

Non-Determinism Our algorithms are not completely deterministic. Hence, the resulting functions and performance numbers may vary slightly. The variance in performance metrics is expected to be negligible.

Synthesized Functions For technical reasons, some of the functions in the provided txt files have extra bits appended that are not from our synthesis algorithm but are added by our scripts. Functions labeled exp5 have a 3-bit slicing function prepended, from the simulated Rocket Lake CPU. Functions labeled exp6 have 5 bits appended to the end, which do not carry meaning in themselves. These bits are overwritten by Champsim by the non-linear slicing function from the simulated Xeon CPU.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.