GreenMalloc: Allocator Optimisation for Industrial Workloads

Aidan Dakhama[1] William B. Langdon[2] Héctor Menéndez[1] Karine Even-Mendoza[1]

[1]King's College London, [2]University College London

aidan@dakhama.com

Introduction

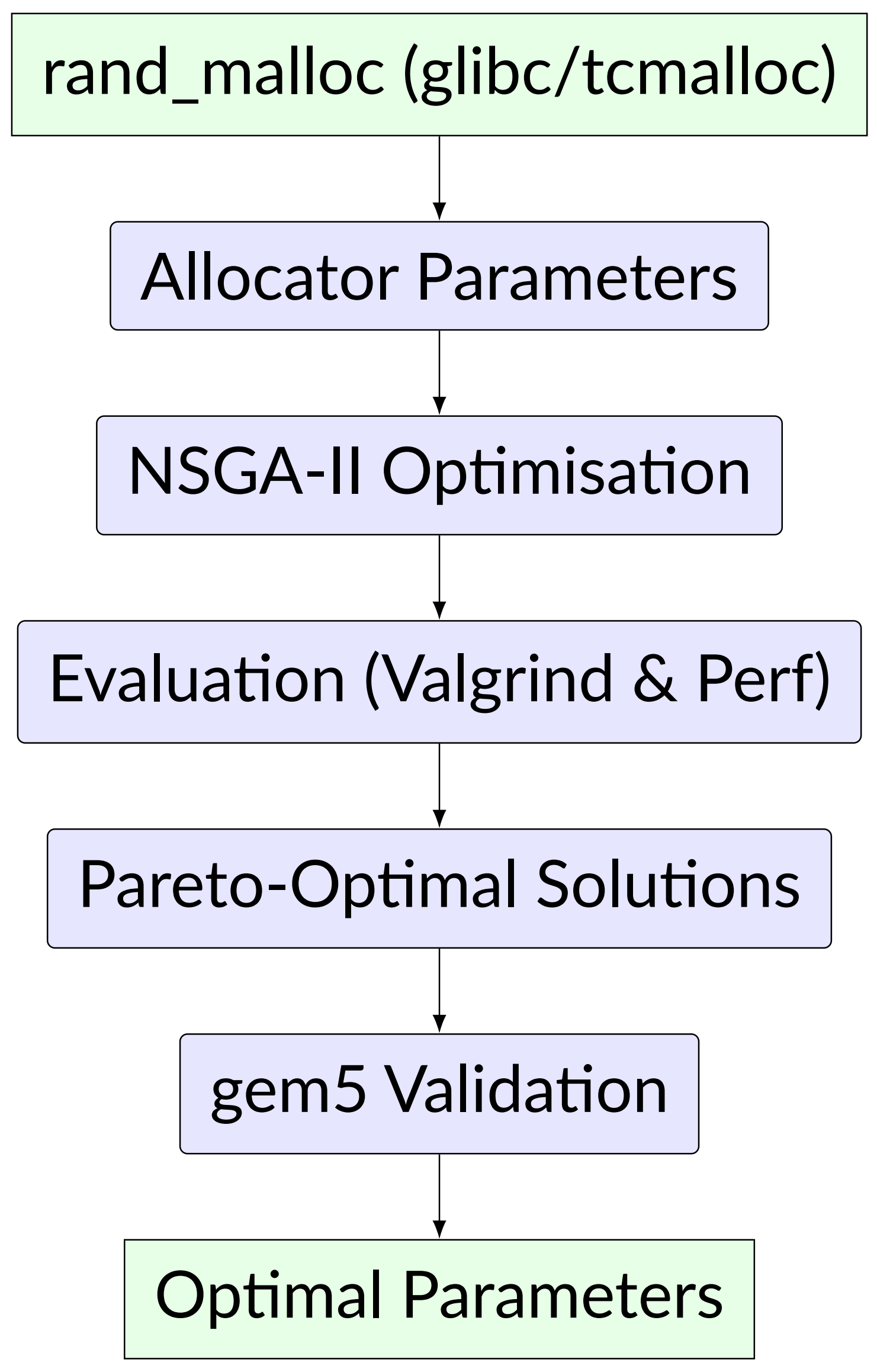
We present GreenMalloc, a multi-objective framework for automatically tuning C++ heap memory allocators. Using NSGA-II and rand_malloc for lightweight benchmarking, we explore allocator parameters and transfer the best configurations to gem5 in a case study of glibc malloc and TCMalloc.

Allocator Optimisation

Efficient memory management is challenging: allocator choice and configuration strongly affect performance. Widely used allocators like glibc malloc and TCMalloc are hard to tune, so systems often use defaults, wasting memory, energy, and performance.

This is worsened in situations where the target system takes a long time to run, and therefore, a long time to benchmark and improve configurations.

Overview of Our Approach



rand_malloc (glibc/tcmalloc)

Allocator Parameters

NSGA-II Optimisation

Evaluation (Valgrind & Perf)

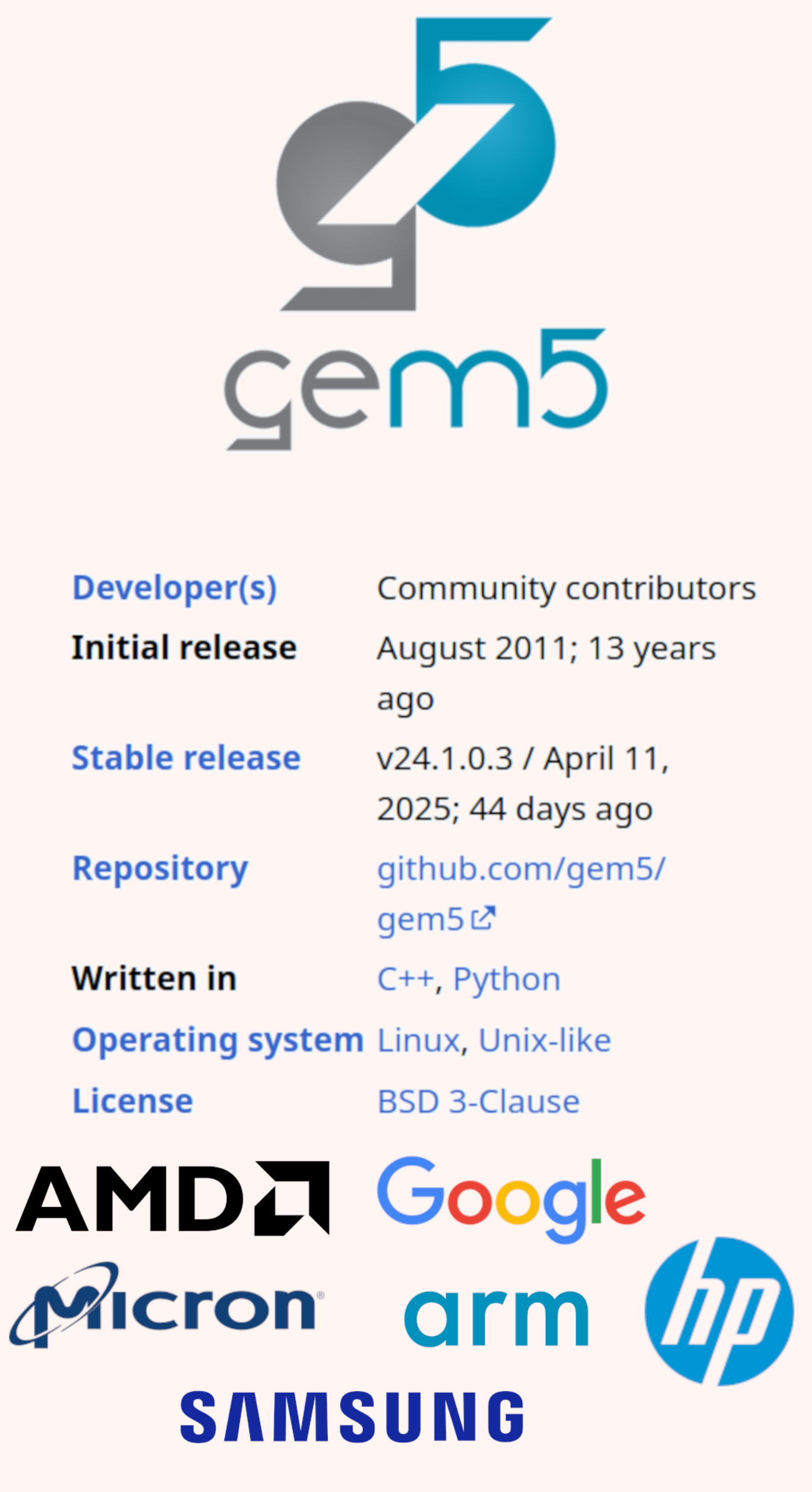
Pareto-Optimal Solutions

gem5 Validation

Optimal Parameters

We begin with rand_malloc, a lightweight benchmark emulating gem5's memory behaviour. Allocators (glibc and tcmalloc) are configured, and a multi-objective genetic algorithm (NSGA-II via pymoo) optimises both peak heap usage and instruction count. The resulting Pareto-optimal configurations are then validated on gem5 to identify the best-performing parameters.

Gem5: Our Case Study



Developer(s)	Community contributors
Initial release	August 2011; 13 years ago
Stable release	v24.1.0.3 / April 11, 2025; 44 days ago
Repository	github.com/gem5/gem5
Written in	C++, Python
Operating system	Linux, Unix-like
License	BSD 3-Clause

Source: Wikipedia (gem5 page, accessed 25/05/2025)

Why Gem5?

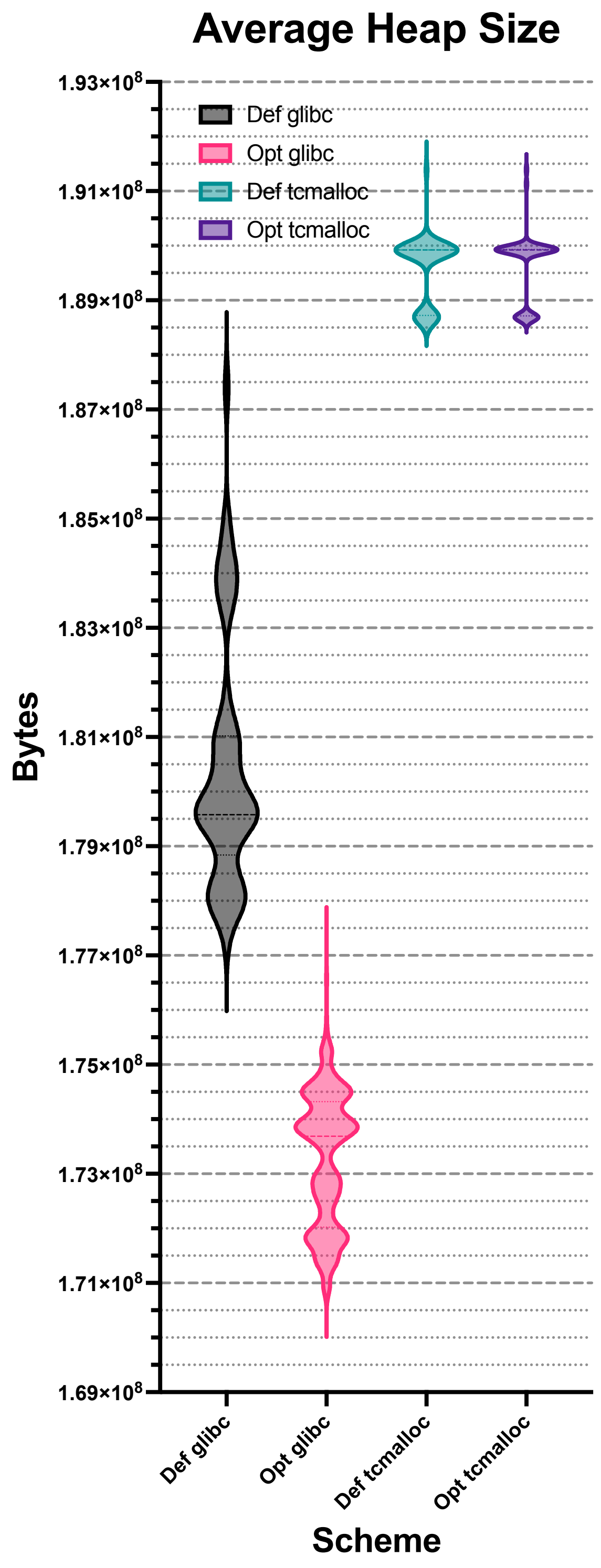
Large: 1.34 million lines of code, reflecting real system complexity

Slow-running: Simulations take substantial time

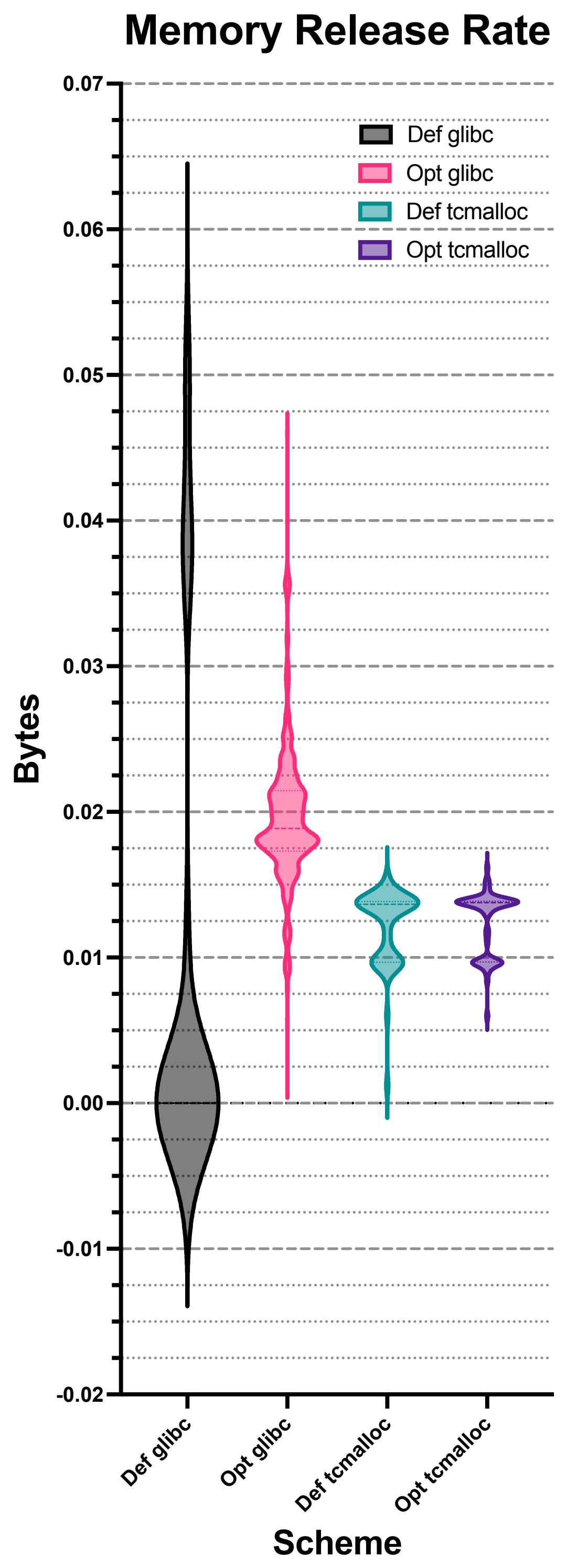
Unusual memory behaviour: Complex allocator interactions and workload-dependent patterns

Industrially relevant: Used for system development, hardware-software co-design, and computer architecture research in both academia and industry

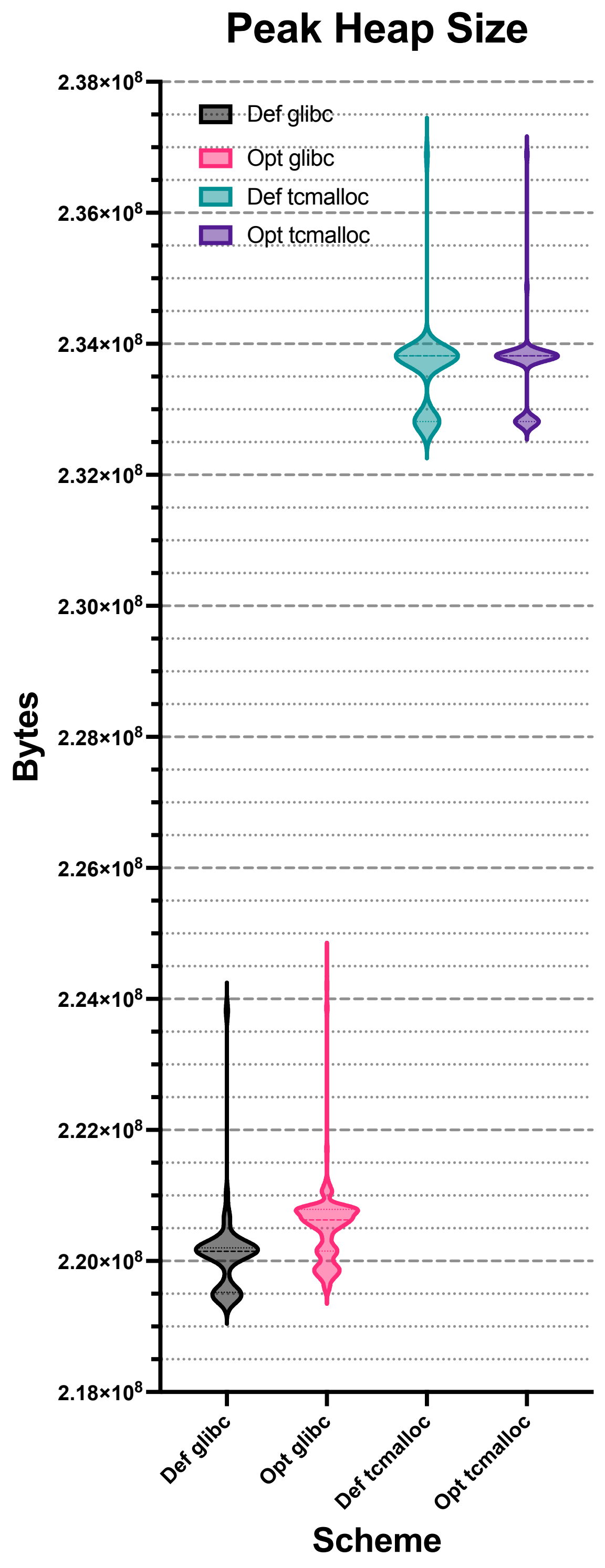
Results



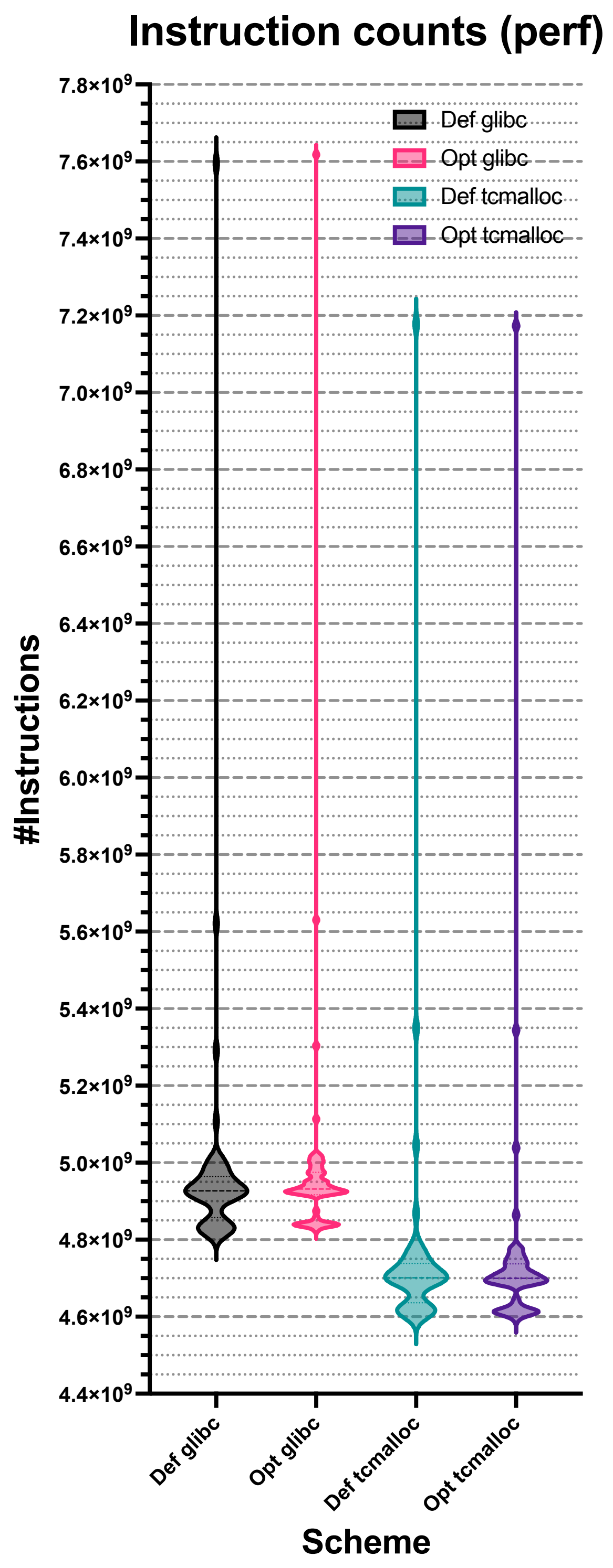
Average Heap Size



Memory Release Rate



Peak Heap Size



Instruction counts (perf)

Allocator trade-offs differ: `glibc` allows gradual tuning with small changes in instructions and heap, while TCMalloc operates near optimal boundaries, offering higher peak-heap potential but requiring steeper trade-offs.

For `glibc`, tuning reduces average memory (180M to 173M) and instructions, while improving memory release. `tcmalloc` shows stable averages but better consistency in memory use and execution. Peak memory remains largely unchanged.

Conclusions

We introduced GreenMalloc, a search-based framework for optimising memory allocator parameters via lightweight benchmarking. On gem5, it reduced average heap usage by up to 4.1% and instruction counts by up to 0.25% across glibc and TCMalloc, demonstrating practical gains in efficiency. This approach can extend to other complex software, including full-system simulation, VMs, and interpreters.

A. Dakhama et al. (2025). "GreenMalloc: Allocator Optimisation for Industrial Workloads". In: *International Symposium on Search Based Software Engineering*. Preprint, under review at SSBSE

W. B. Langdon (2025). "A Genetic Improvement Parameter Benchmark: rand_malloc.c". In: *UKCI*. URL: `https://gpbib.cs.ucl.ac.uk/gp-html/Langdon_2025_UKCI.html`

N. Binkert et al. (2011). "The gem5 Simulator". In: *SIGARCH Comput. Archit. News* 39.2, pp. 1–7. ISSN: 0163-5964. DOI: `10.1145/2024716.2024718`