AUTOMATED ALGORITHM DESIGN

For Auto-Tuning Optimizers

Floris-Jan Willemsen, Niki van Stein & Ben van Werkhoven | MLSys 2026

Universiteit
Leiden
The Netherlands

What is auto-tuning?

- In Computer Science:

  "Automatic optimization of those parameters, which are susceptible of being adapted, for a specific problem to be solved in a given architecture." [1]

  - Common types of objectives:

    - performance
    - energy efficiency
    - accuracy

[1] Developing Linear Algebra Codes on Modern Processors: Emerging Research and Opportunities, Chapter 5, Pallarés et al., 2023

Supercomputer lifetimes

Average time until succeeded: 5.2 years

Jaguar
2009

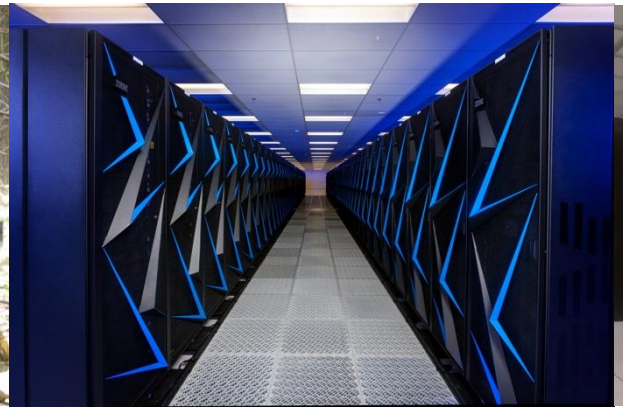
Titan
2012

Summit
2018

Frontier
2022

Discovery
2028

Sequoia
2012

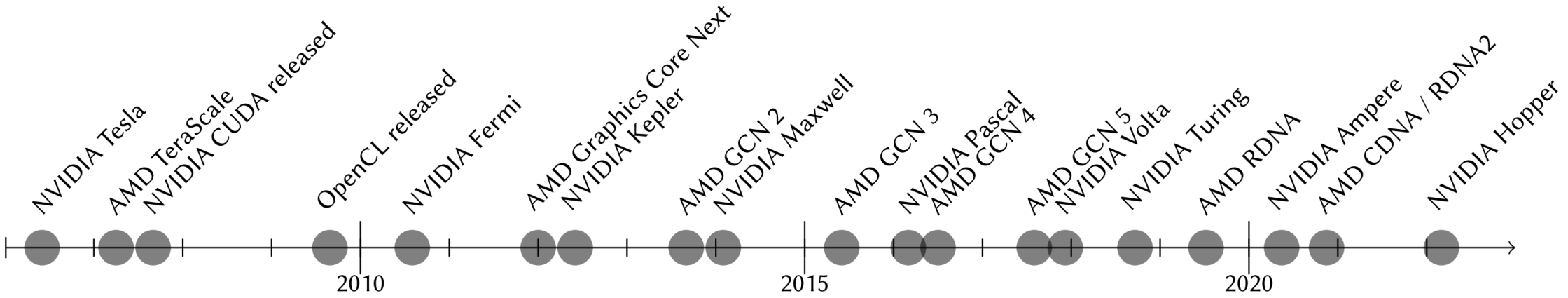


Sierra
2018

El Capitan
2024

GPU architecture lifetimes

Average lifetime: 1.96 years



Supercomputer application lifetimes

Some widely-used applications[1]:

Application	area	Initial release	Latest release
VASP	Atomic-scale materials	1989	2025
LAMMPS	Atomic/molecular simulation	1995	2024
cp2k	Quantum chemistry & solid-state physics	2000	2025
GROMACS	Molecular dynamics simulations	1991	2025
NEMO	Ocean circulation model	1998 (components dating back to 1980s)	2025

- Average age: 30.2 years

[1] according to Archer2 usage data, release dates on Wikipedia and nemo-ocean.eu

Sustainability and productivity problem

Achieving high-performance requires optimizing the code to efficiently use the underlying hardware

Problem:

- How can applications adapt to new hardware every 1-5 years?
- How not to solve it:

```
#if __CUDA_ARCH__ == 700 || __CUDA_ARCH__ == 720
  unsigned statY    = firstBlockY + statYoffset + NR_BLOCKS_PER_TCM_Y * y + ((threadIdx.x >> 3) & 2) + (threadIdx.x & 4);
  unsigned statX    = firstBlockX + statXoffset + NR_BLOCKS_PER_TCM_X * x + ((threadIdx.x >> 2) & 2);
  unsigned polY     = threadIdx.x & 1;
  unsigned polX     = (threadIdx.x >> 1) & 1;
#elif __CUDA_ARCH__ == 750 || __CUDA_ARCH__ == 800 || __CUDA_ARCH__ == 860
  unsigned statY    = firstBlockY + statYoffset + NR_BLOCKS_PER_TCM_Y * y + ((threadIdx.x >> 3) & 3);
  unsigned statX    = firstBlockX + statXoffset + NR_BLOCKS_PER_TCM_X * x + ((threadIdx.x >> 1) & 1);
  unsigned polY     = (threadIdx.x >> 2) & 1;
  unsigned polX     = threadIdx.x & 1;
#endif
```

Solution: tunable code

Parametrize the code:

- based on implementation choices, not architecture features
- without hard coding constants in the source code

To maximize GPU code performance, you need to find the best combination of:

- Different mappings of the problem to threads and thread blocks
- Different data layouts in different memories (shared, constant, …)
- Different ways of exploiting special hardware features
- Application-specific optimizations that may be applied to various degrees
- Work per thread in each dimension
- Loop unrolling factors
- Overlapping computation and communication

Solution: have an auto-tuning framework do it for you!

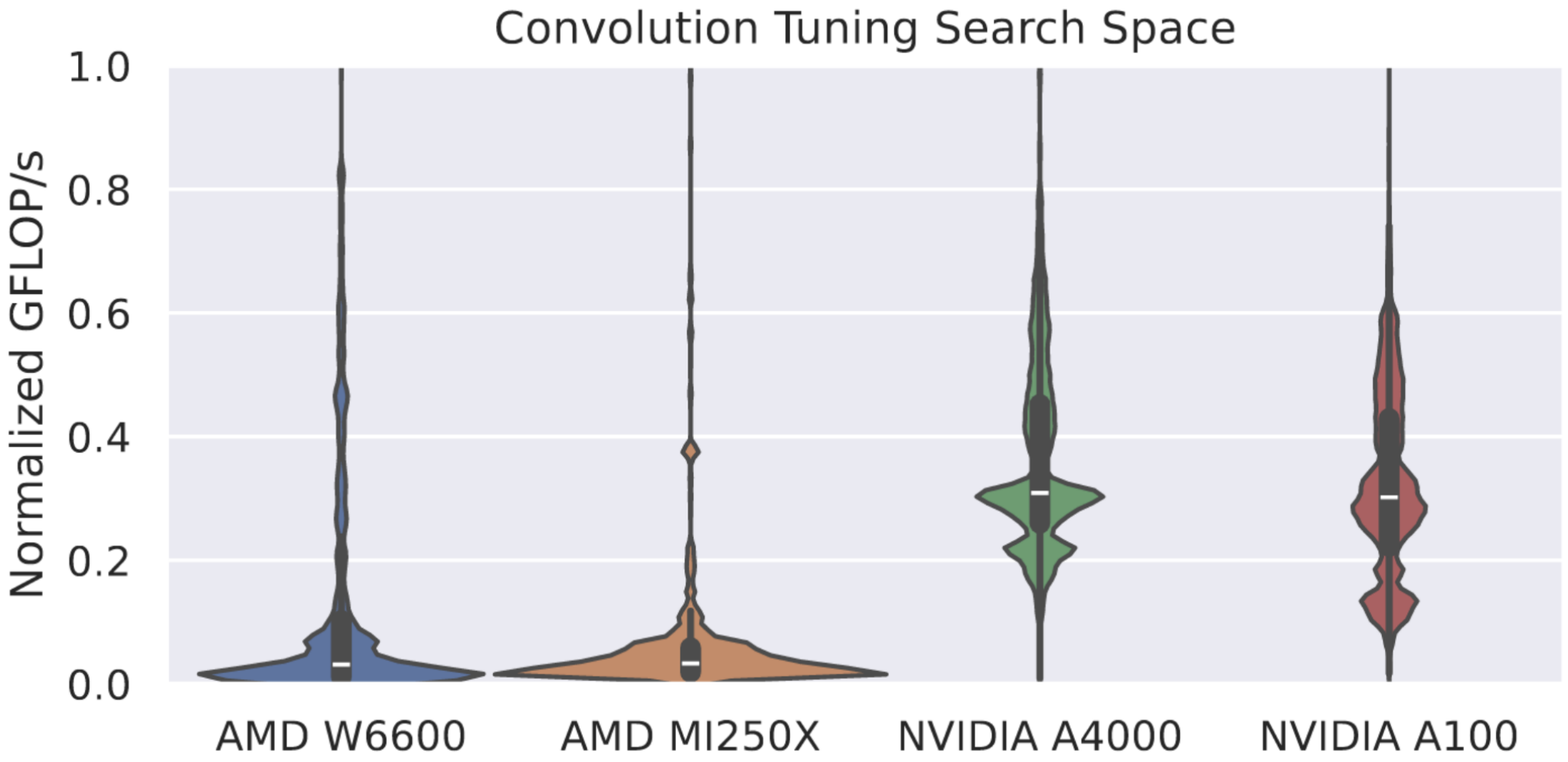
Kernel Tuner – *A Python tool for auto-tuning GPU kernels*

- Started in 2016, now developed by the Kernel Tuner research team
- Funded by several national and European projects
- Completely free and open source

- Supports many different use cases:
  - Any host programming language
  - CUDA, HIP, OpenCL, OpenACC, C, Fortran…
  - Verification
  - Accuracy tuning
  - Energy tuning
  - Observers
- Highly modular design
- All in an accessible Python package

Kernel Tuner research team

What does auto-tuning get you?



[2] Bringing Auto-Tuning to HIP: Analysis of Tuning Impact and Difficulty on AMD and Nvidia GPUs, Luratie et al., EuroPAR 2024

Auto-Tuning needs Optimization Algorithms

- Auto-tuning is essential for complex high-performance applications
- Auto-tuning requires good optimization algorithms to be efficient
- Auto-tuning search spaces are large, discontinuous, and irregular
- Human-designed optimizers are more general and require manual engineering
- Research gap: why are there no optimizers designed for auto-tuning problems?
  - Generalization across applications and hardware is hard
  - Designing an algorithm takes time
  - Repeated auto-tuning across a representative set of applications and hardware is infeasible

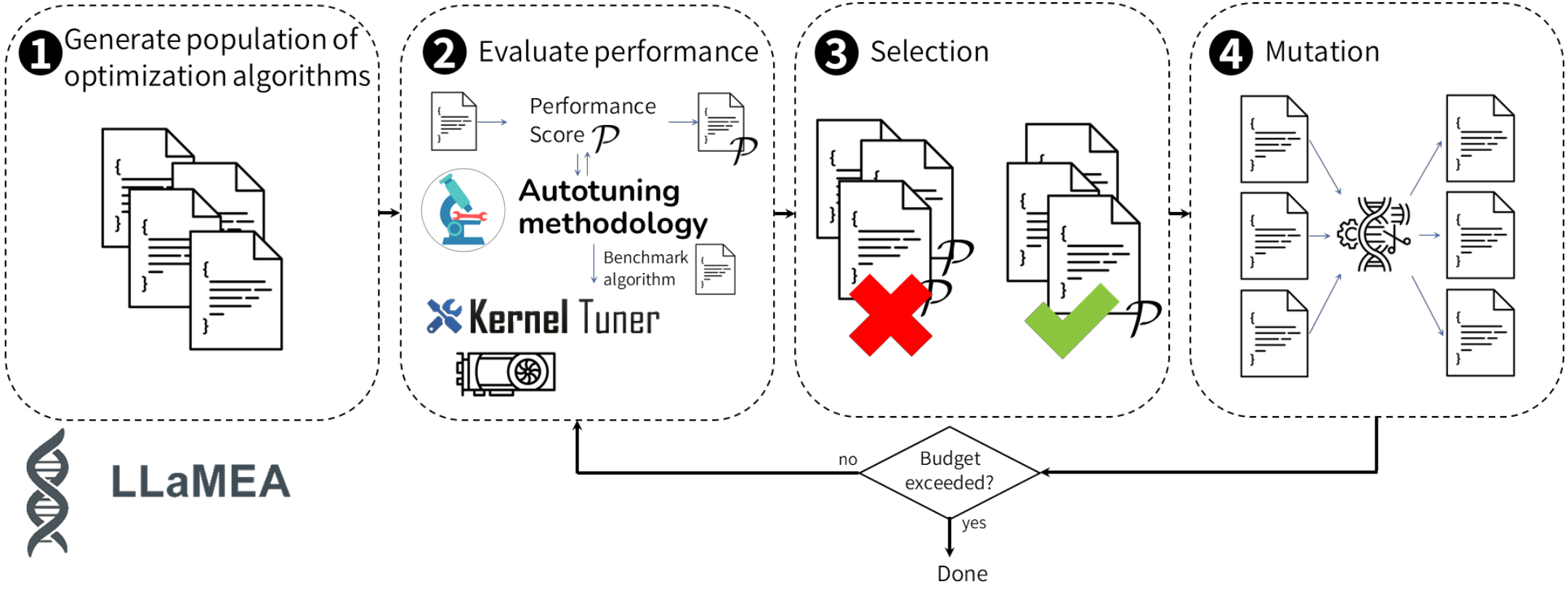
Three Enabling Advancements

- Generalization across applications and hardware: Autotuning Methodology [3]
  - Performance score P measures area under the performance-vs-budget curve
  - Captures both convergence speed and final solution quality
- Repeated auto-tuning across a representative set of applications and hardware: simulation mode [4]
- Manual engineering replaced with LLMs and feedback loop: LLaMEA [5]

[3] A methodology for comparing optimization algorithms for auto-tuning, Willemsen et al., FGCS 2024
[4] Tuning the Tuner: Introducing Hyperparameter Optimization for Auto-Tuning, Willemsen et al., IEEE eScience 2025
[5] LLaMEA: a large language model evolutionary algorithm for automatically generating metaheuristics, van Stein and Bäck 2025

Automated Design Loop

- Kernel Tuner optimizes tuning problems
- LLaMEA generates and mutates optimizers
- Autotuning Methodology provides evaluation metric



Evaluation

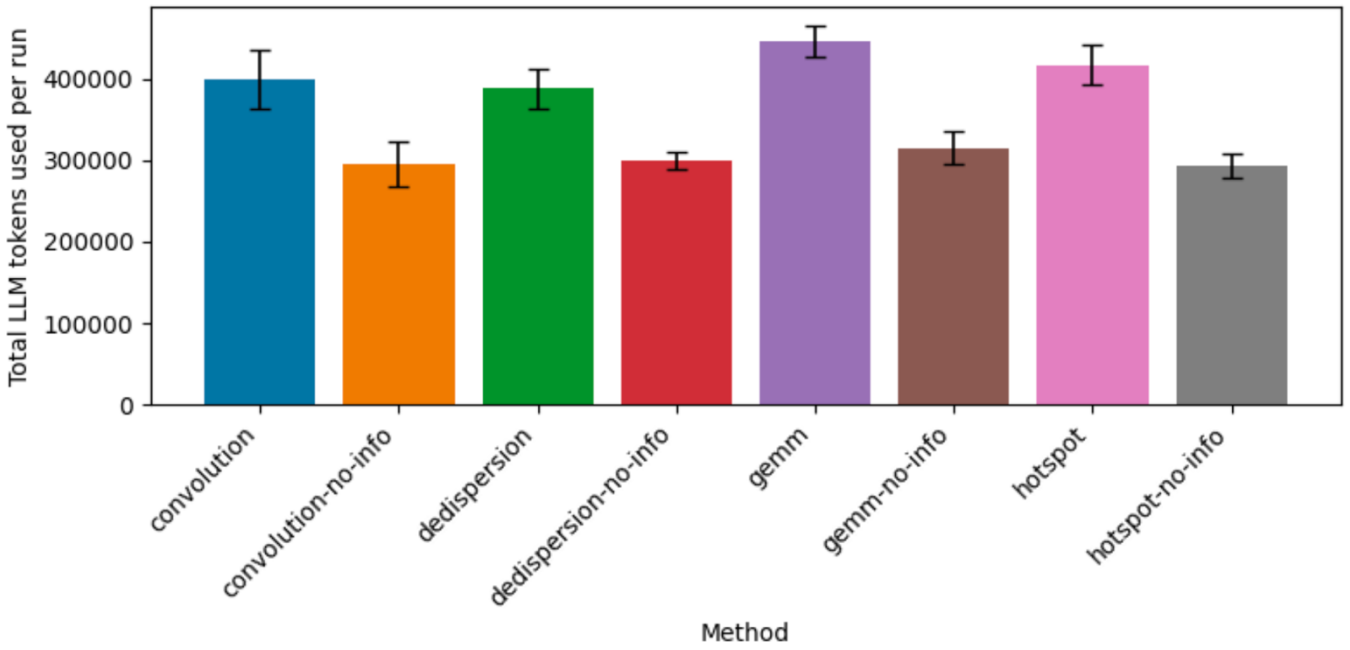
- 4 kernels: GEMM, Hotspot, Dedispersion, Convolution
- 6 GPUs:
  - AMD MI250X
  - AMD W6600
  - AMD W7800
  - Nvidia A100
  - Nvidia A4000
  - Nvidia A6000
- Results in 24 search spaces
- Feasible due to simulation mode

Name	Cartesian size	Constrained size	Dimensions	No. constraints	No. values per parameter	Density %
Dedispersion	22272	11130	8	3	1 - 29	49.973
2D Convolution	10240	4362	10	4	1 - 16	42.598
Hotspot	22200000	349853	11	5	1 - 37	1.576
GEMM	663552	116928	17	8	1 - 4	17.622

GPU	Year	Architecture	Cores	Memory	Cache	Bandwidth (GB/s)	Peak SP (GFLOPS/s)
AMD W6600	2021	RDNA 2	1792	16 GB GDDR6	32 MB L3	224	10404
AMD MI250X*	2021	CDNA 2	7040	64 GB HMB2e	8 MB L2	1638	28160
AMD W7800	2023	RDNA 3	4480	32 GB GDDR6	64 MB L3	576	45250
Nvidia A4000	2021	Ampere	6144	8 GB GDDR6	4 MB L2	448	17800
Nvidia A6000	2020	Ampere	10752	48 GB GDDR6	6 MB L2	768	38710
Nvidia A100	2020	Ampere	6912	40 GB HMB2	40 MB L2	1555	19500

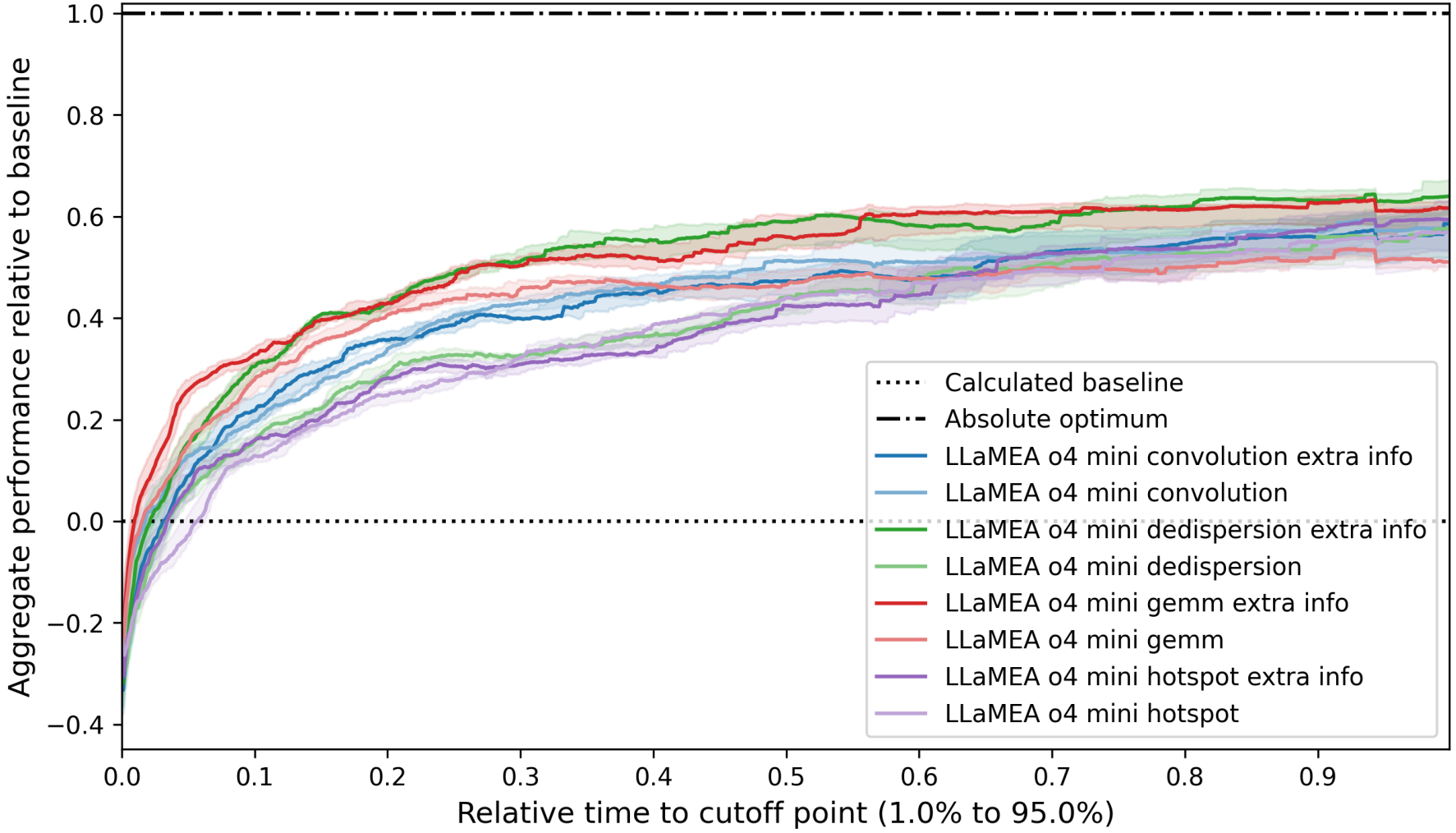
Evaluation

- For each kernel, two experiments were set up:
  - Without search space information: provided solely with an auto-tuning task description.
  - With search space information: provided with information on possible parameter values and constraints.
  - For each experiment, 5 independent runs were made, 100 LLM calls each, best out of five selected
  - Executed on OpenAI 04-mini



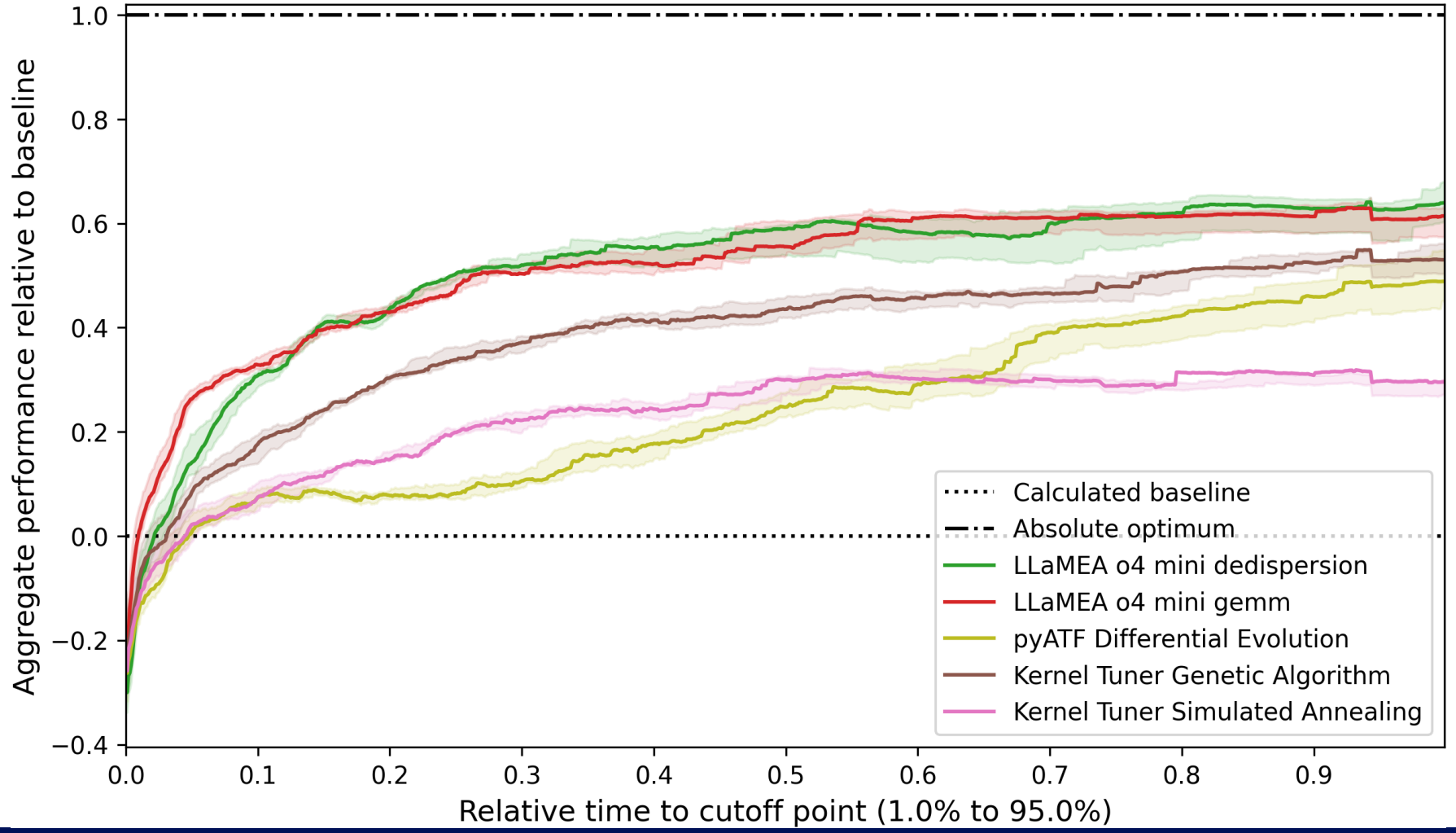
Results: aggregate

- Providing extra information improves performance by 30.7%



Results: versus state-of-the-art

- Against the best hyperparameter-tuned human-designed optimization algorithms: average improvement of 72.4%



Conclusions

- LLMs can generate auto-tuning optimizers surpassing optimized human-designed algorithms
- More powerful LLMs likely to yield even better results
- Methods are integrated into the open-source LLaMEA and Kernel Tuner frameworks
  - pip install llamea
  - pip install kernel_tuner

AUTOMATED ALGORITHM DESIGN

For Auto-Tuning Optimizers

Floris-Jan Willemsen, Niki van Stein & Ben van Werkhoven | MLSys 2026

Universiteit
Leiden
The Netherlands