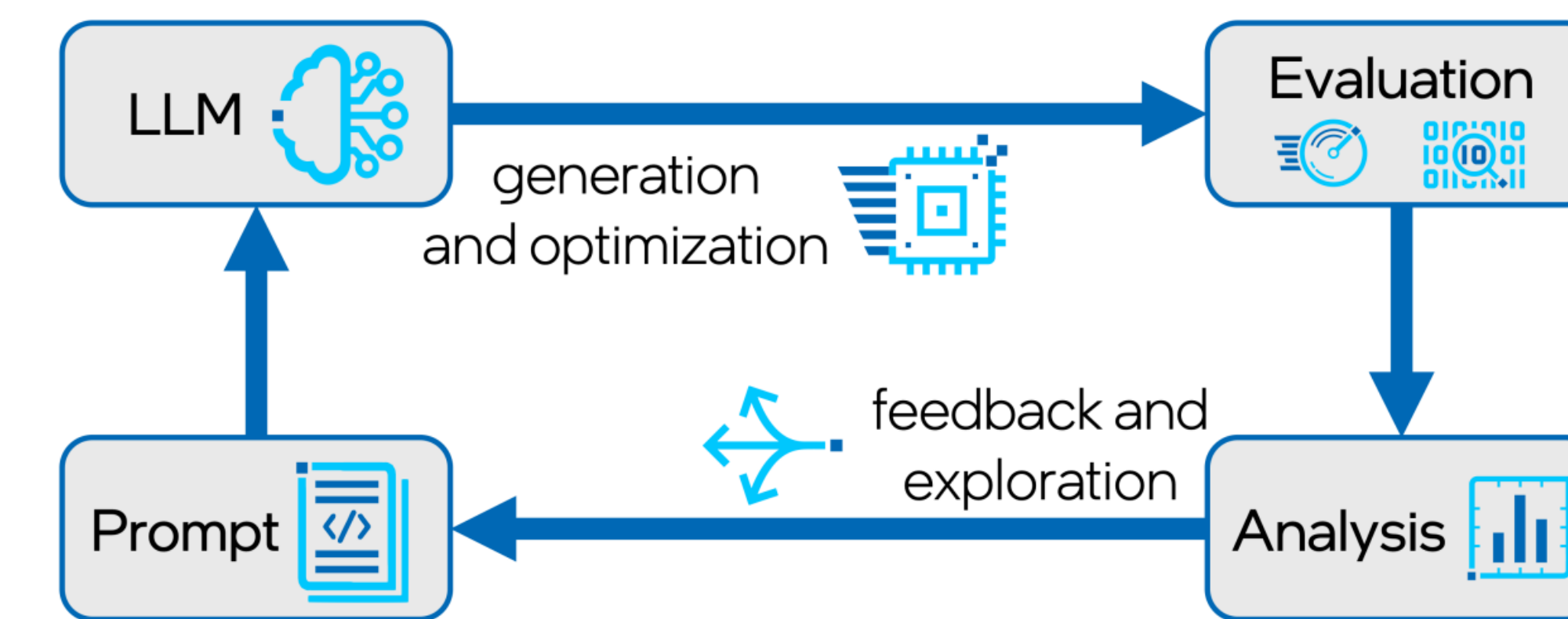


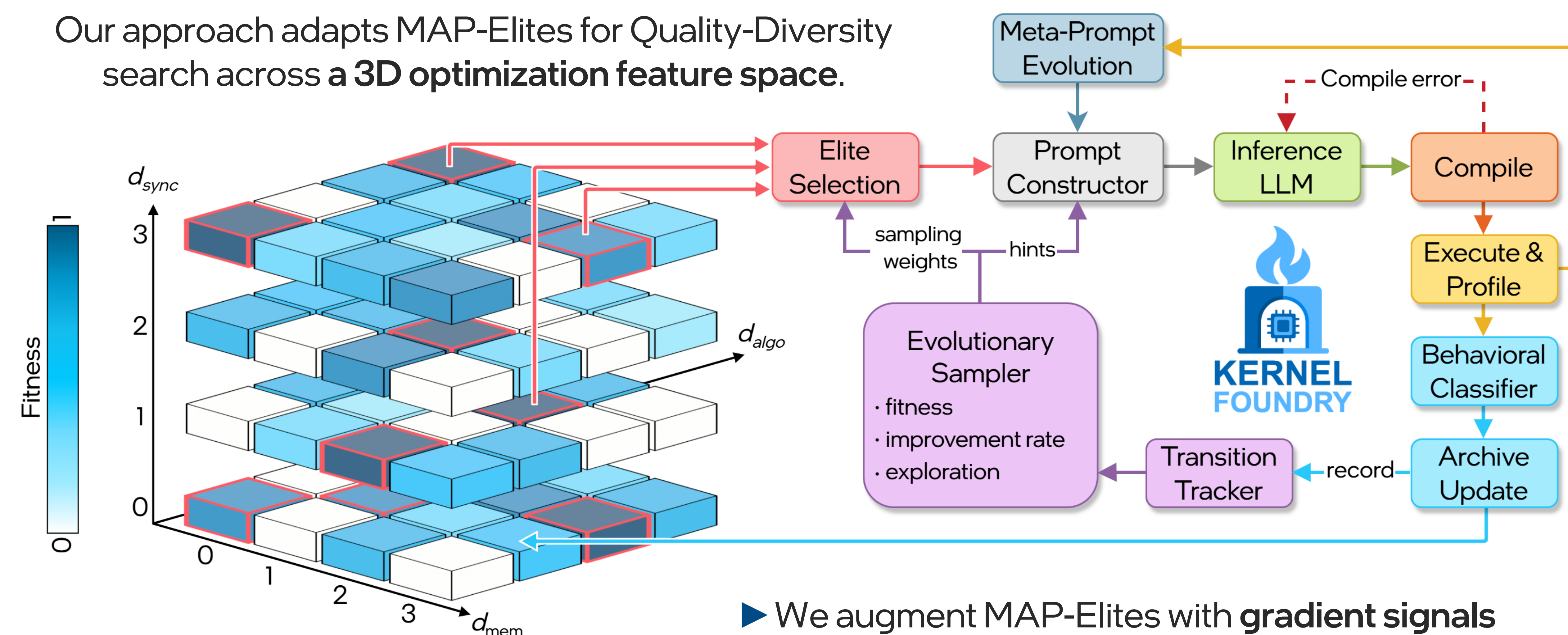
Efficient GPU kernels are critical for performant AI models.

- ▶ **High-performance GPU kernel engineering** requires deep expertise in hardware architectures, memory hierarchies, and parallel programming.
- ▶ **Prior work** uses iterative generate → verify → measure loops, focusing on CUDA and Triton.
- ▶ **Limitations** include local optima and no systematic exp



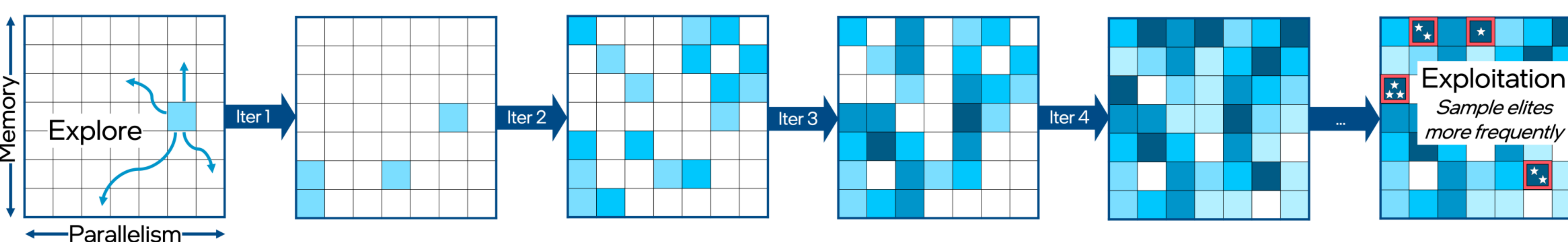
We propose an evolutionary search method for GPU kernel optimization.

Our approach adapts MAP-Elites for Quality-Diversity search across a **3D optimization feature space**.



	Memory	Algorithm	Parallelism
0	scalar, strided, uncoalesced	direct PyTorch translation	no sync
1	coalesced, vectorized	fused ops	work-group barriers
2	shared/local mem. with tiling	reformulated algorithm	sub-group primitives
3	multi-level hierarchy	novel, improved asymptotically	global coord. (e.g., atomics)

- ▶ We augment MAP-Elites with **gradient signals** from transition history. These inform the sampling weights in parent selection and mutation hints in prompt construction to help **steer exploration**.
- ▶ We allow prompts to co-evolve with kernels via a dedicated **Meta-Prompter** that diagnoses and **refines guidance**, e.g., a “common pitfalls” section.
- ▶ We enable explicit HW-specific **parameter tuning** through configurable **templated kernels** that can be dispatched and evaluated in a sweep.

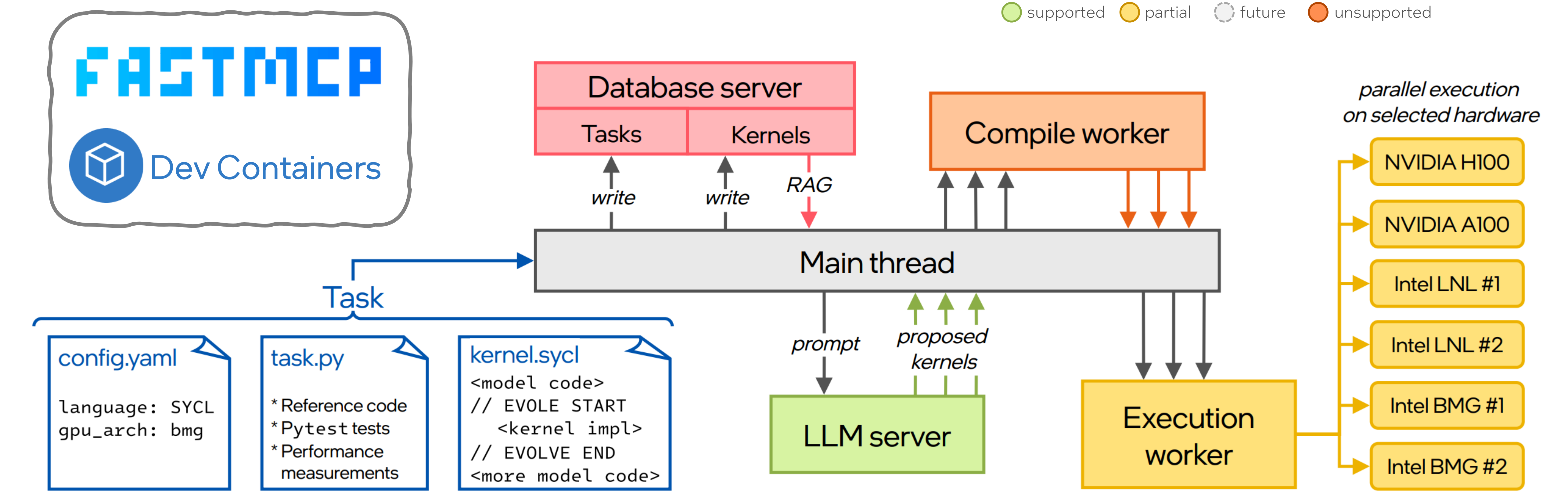


Our method is designed to be scalable and developer-friendly.

- ▶ **Customizable task** format using pytest
- ▶ **IDE integration** as KernelFoundry agent and MCP server
- ▶ One-click setup with **dev containers**
- ▶ Broad support for **cross-vendor languages/DSLs**
- ▶ Infrastructure workers connected via a **load balancer** and **queue** system

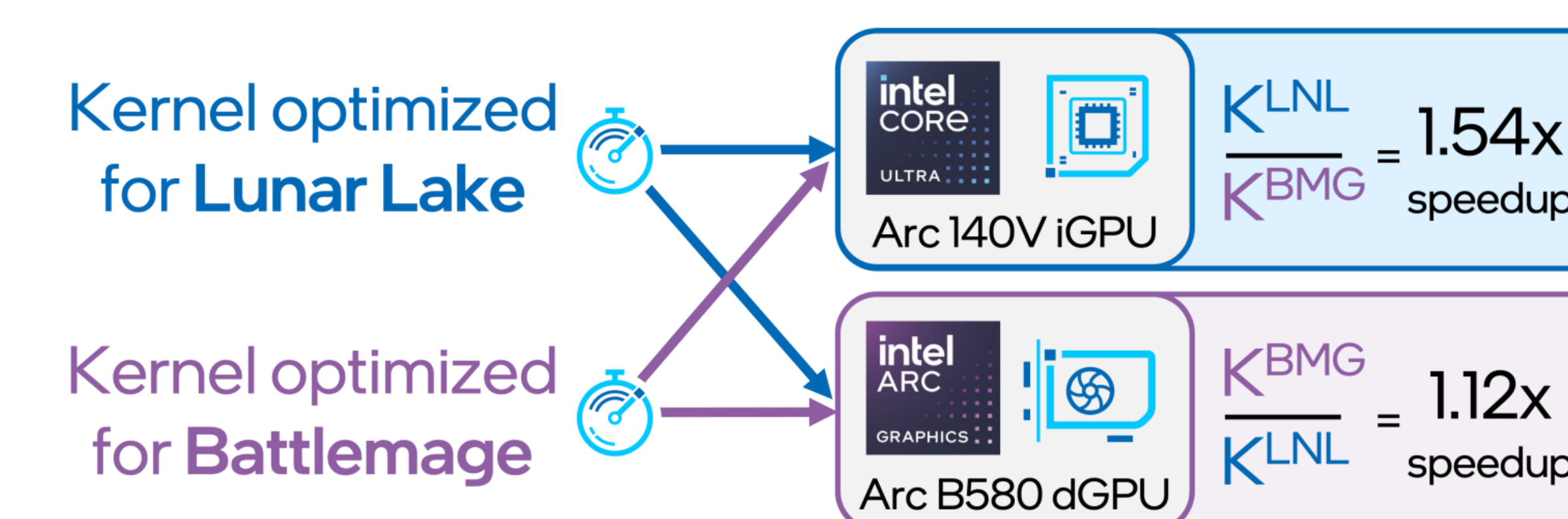
Language / DSL	Intel GPUs	Nvidia GPUs
SYCL ISO C++ / oneAPI DPC++		
TRTN Python DSL / PyTorch		
CL C99 / C++ Compute Standard		
CUDA NVIDIA Native C++		
VK Compute Shaders		

 supported
 partial
 future
 unsupported

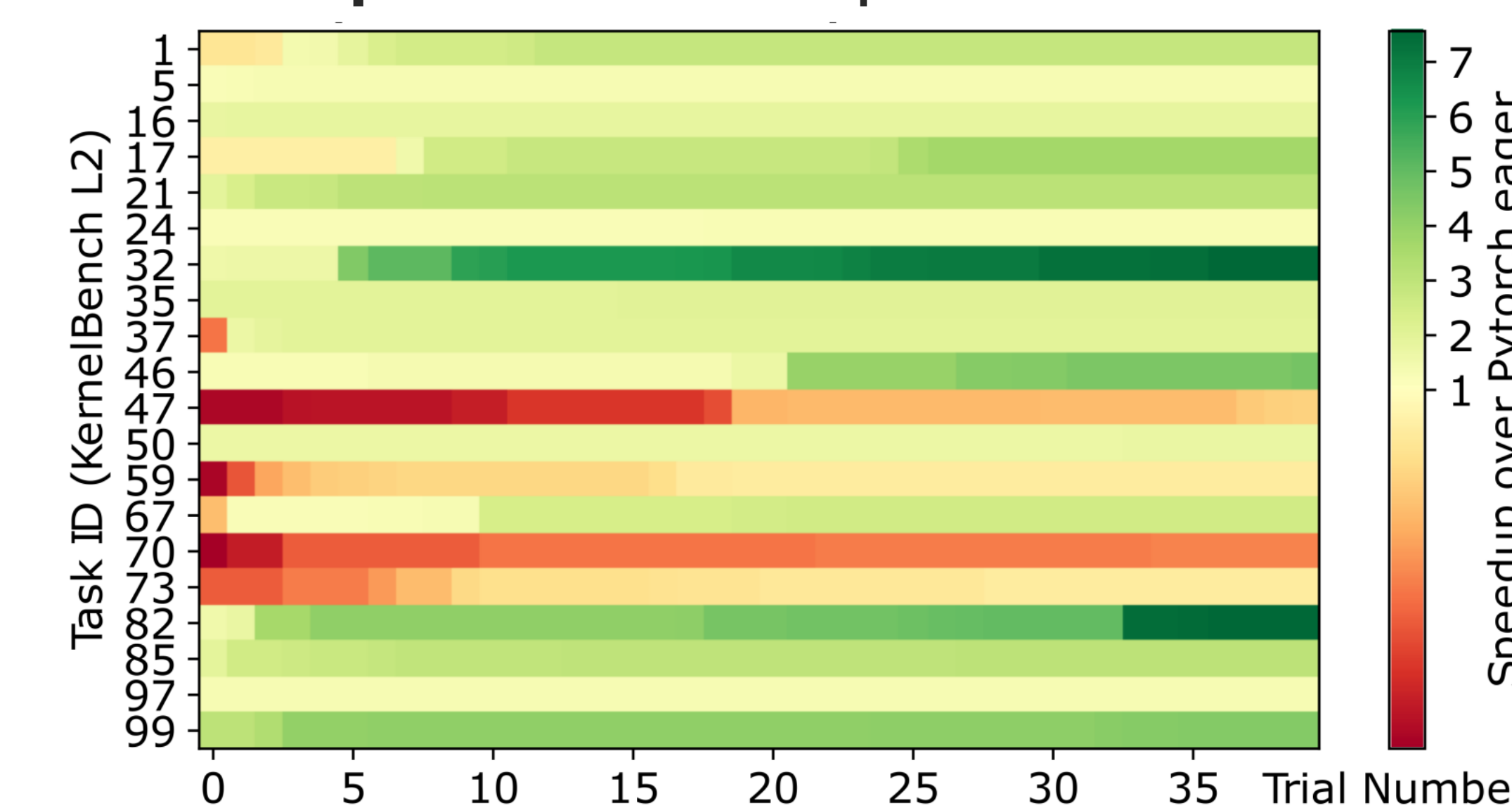


KernelFoundry excels across benchmarks, backends, and HW platforms.

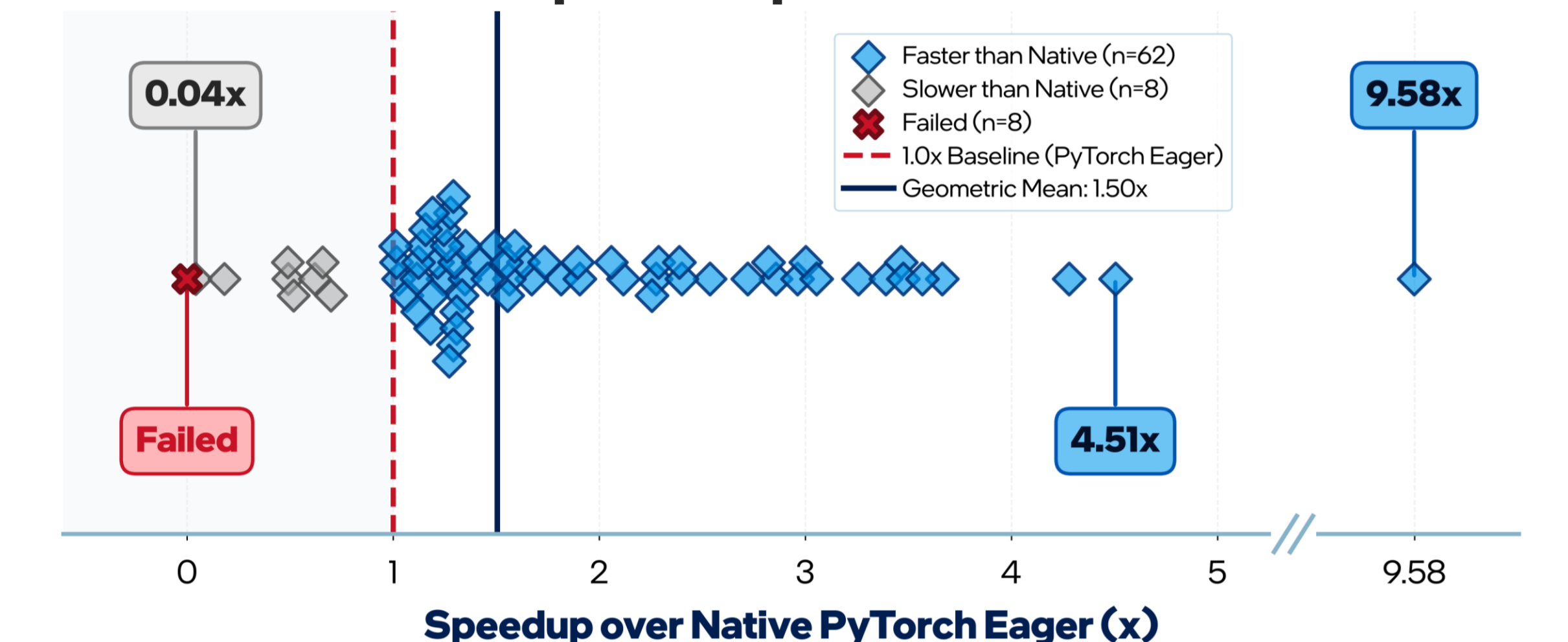
Our approach produces **hardware-aware** kernels.



Our method **progressively improves** kernel performance.



Triton kernel speedups on KernelBench L2



We demonstrate the success of our evolutionary algorithm on **SYCL kernel generation**.

Method	# Iter.	Avg. Speedup
OpenEvolve	10	1.483
<i>(open-source impl. of AlphaEvolve)</i>	40	2.535
Ours (with kernel-specific optim. strategies, meta-prompting, and gradient guidance)	10	2.059
	20	2.539
Ours + parameter optimization	40	2.732

Task set: KernelBench L2 (n=20). All methods use the same LLM ensemble.