

PyTorch Conference 2026

Making vLLM Faster on Intel GPUs with Triton Kernels

Whitney Tsang, Principal Engineer, Intel

Artur Fierka, AI Software Solution Engineer, Intel



Notices and Disclaimers

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](https://www.intel.com/PerformanceIndex).

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Results have been estimated or simulated.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

All product plans and roadmaps are subject to change without notice.

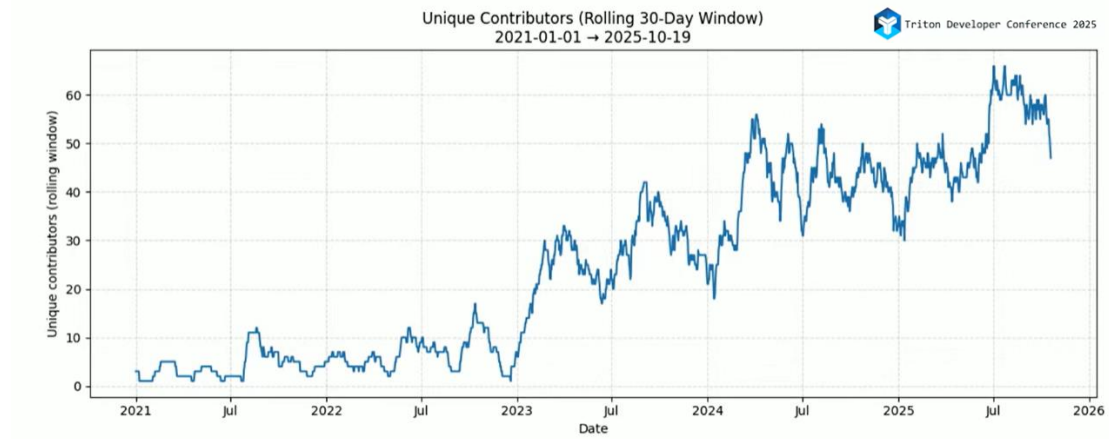
Statements in this document that refer to future plans or expectations are forward-looking statements. These statements are based on current expectations and involve many risks and uncertainties that could cause actual results to differ materially from those expressed or implied in such statements. For more information on the factors that could cause actual results to differ materially, see our most recent earnings release and SEC filings at www.intc.com.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Triton's Ecosystem Keeps Growing

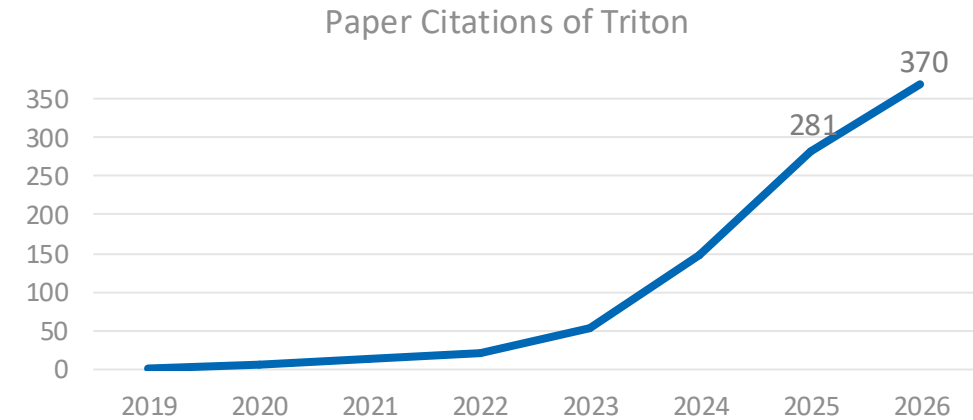
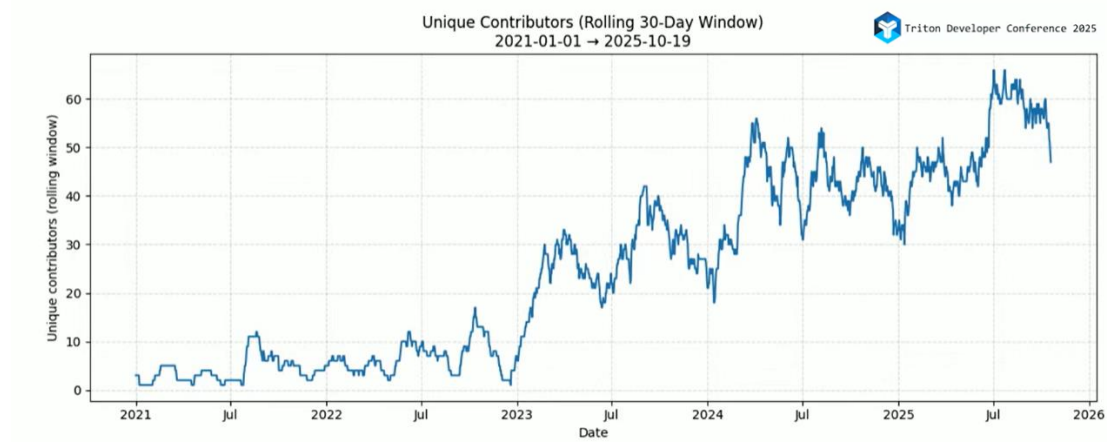
More contributors, more research, more production inference



Sources: [Triton's 2019 paper](#), [OpenAI Triton's announcement](#), [Semantic Scholar](#)

Triton's Ecosystem Keeps Growing

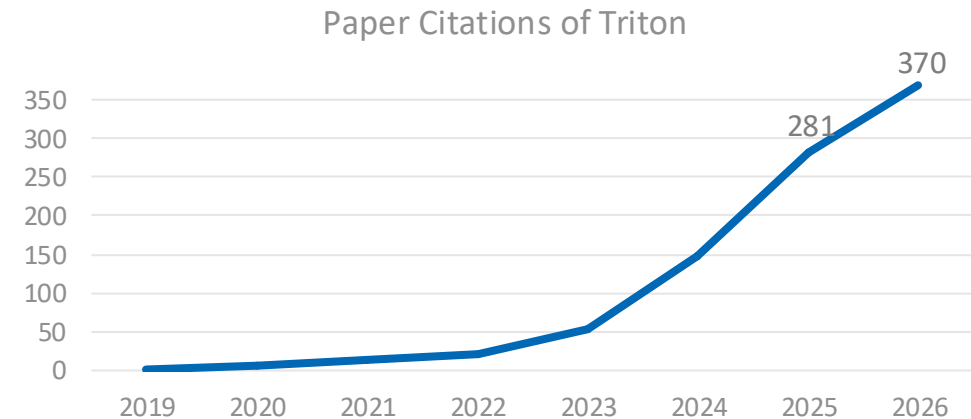
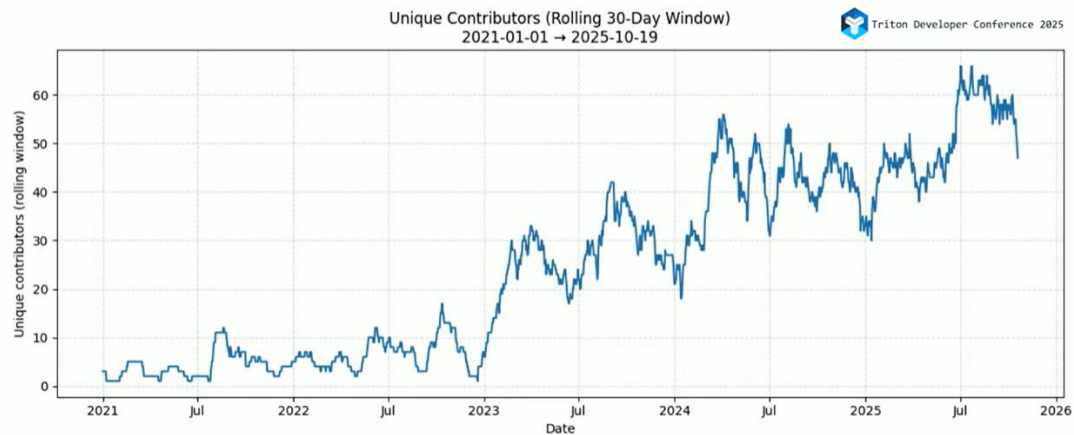
More contributors, more research, more production inference



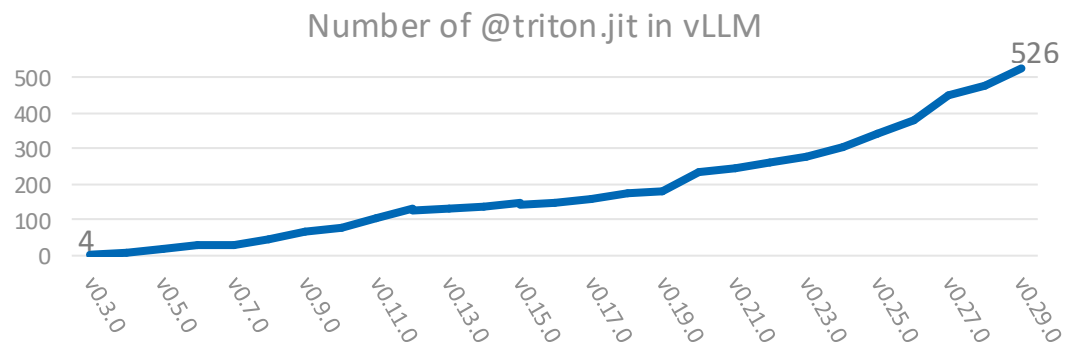
Sources: [Triton's 2019 paper](#), [OpenAI Triton's announcement](#), [Semantic Scholar](#)

Triton's Ecosystem Keeps Growing

More contributors, more research, more production inference



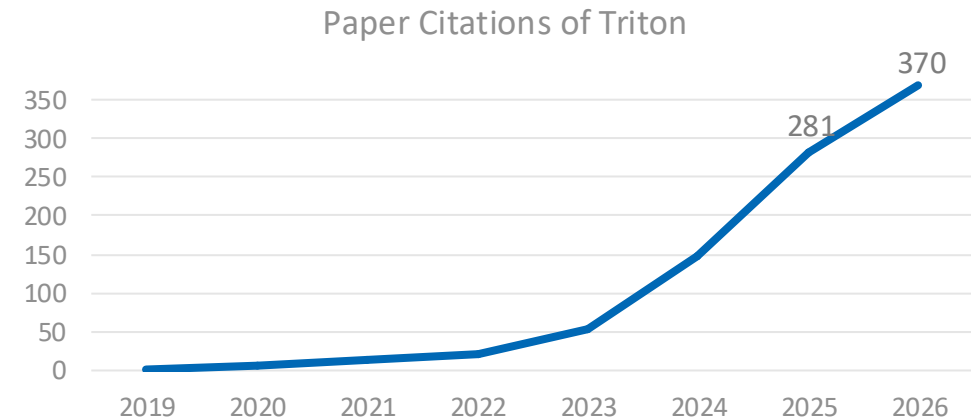
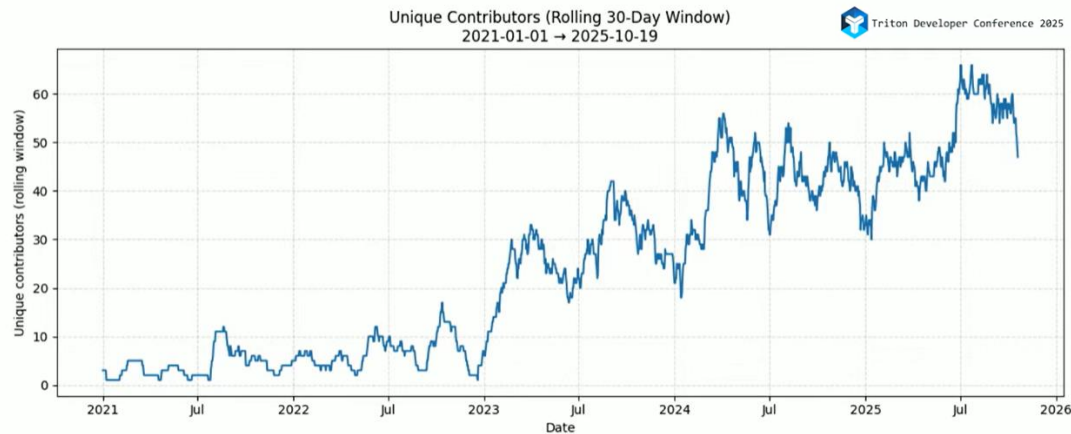
Explosive Growth of Triton Kernels in vLLM



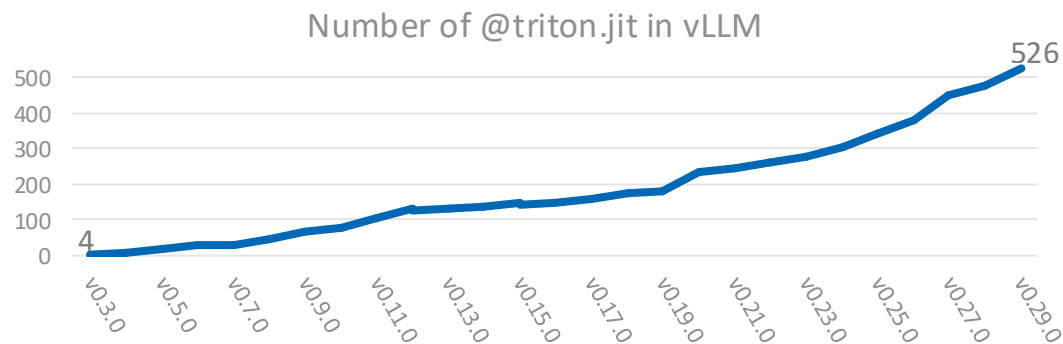
Sources: [Triton's 2019 paper](#), [OpenAI Triton's announcement](#), [Semantic Scholar](#)

Triton's Ecosystem Keeps Growing

More contributors, more research, more production inference



Explosive Growth of Triton Kernels in vLLM



And this is only the explicit kernels — every torch.compile'd model also lowers to Triton via Inductor.

Sources: [Triton's 2019 paper](#), [OpenAI Triton's announcement](#), [Semantic Scholar](#)

95%+ Performance with 70x Less Code

GPU Performance Portability Needs Autotuning

Demonstrating portable and performant cross-platform LLM kernels

Burkhard Ringlein[📧], Thomas Parnell[📧], and Radu Stoica[📧]

The Anatomy of a Triton Attention Kernel

How to achieve cross-platform state-of-the-art LLM attention using only Triton

Burkhard Ringlein, Jan van Lunteren, Radu Stoica, Thomas Parnell

IBM Research

Zurich, Switzerland

ngl, jvl, rst, tpa@zurich.ibm.com

vLLM Triton Attention Backend Deep Dive

March 4, 2026 10 min read

vLLM Team at IBM Research

Sources: [arXiv: 2505.03780](#), [arXiv: 2511.11581](#), [vLLM blog - Triton article](#)

95%+ Performance with 70x Less Code

GPU Performance Portability Needs Autotuning

Demonstrating portable and performant cross-platform LLM kernels

Burkhard Ringlein, Thomas Parnell, and Radu Stoica

The Anatomy of a Triton Attention Kernel

How to achieve cross-platform state-of-the-art LLM attention using only Triton

Burkhard Ringlein, Jan van Lunteren, Radu Stoica, Thomas Parnell
IBM Research
Zurich, Switzerland
ngl, jvl, rst, tpa@zurich.ibm.com

vLLM Triton Attention Backend Deep Dive

March 4, 2026 10 min read
vLLM Team at IBM Research

TABLE I
INVESTIGATED LLM KERNEL IMPLEMENTATIONS

	Implementation	LoC	Target vendor	Source
Attention	flash_attn	69197	NVIDIA	github.com/Dao-AI/flash-attention, [1], [10]
	rocm_flash_attn	52489	AMD	github.com/ROCm/flash-attention
	pytorch native	29	NVIDIA / AMD	pytorch/.../functional.py, [6]
	Triton manual	1049	NVIDIA / AMD	[11]
	Triton w/ autotuning	1100	NVIDIA / AMD	ibm.biz/vllm-ibm-triton-lib (this work)
RMS norm	layernorm_kernels.cu	159	NVIDIA (& AMD via hipify)	github.com/vllm-project/vllm, [3]
	Triton w/ autotuning	96	AMD / NVIDIA	ibm.biz/vllm-ibm-triton-lib (this work)

Sources: [arXiv: 2505.03780](#), [arXiv: 2511.11581](#) [vLLM blog - Triton article](#)

95%+ Performance with 70x Less Code

GPU Performance Portability Needs Autotuning

Demonstrating portable and performant cross-platform LLM kernels

Burkhard Ringlein[✉], Thomas Parnell[✉], and Radu Stoica[✉]

The Anatomy of a Triton Attention Kernel

How to achieve cross-platform state-of-the-art LLM attention using only Triton

Burkhard Ringlein, Jan van Lunteren, Radu Stoica, Thomas Parnell
IBM Research
Zurich, Switzerland
njl, jvl, rst, tpa@zurich.ibm.com

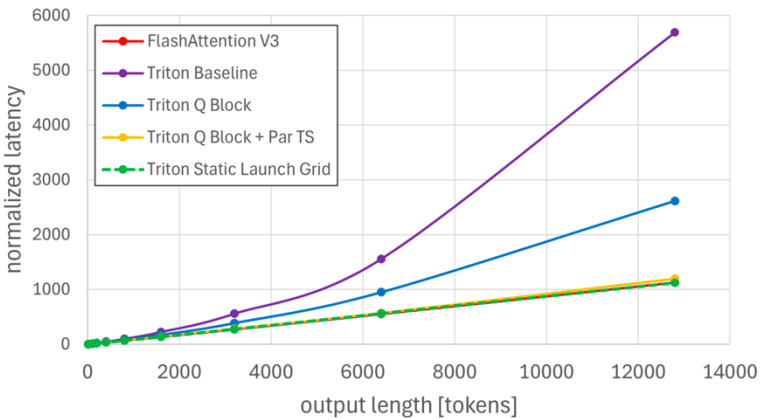
vLLM Triton Attention Backend Deep Dive

March 4, 2026 10 min read
vLLM Team at IBM Research

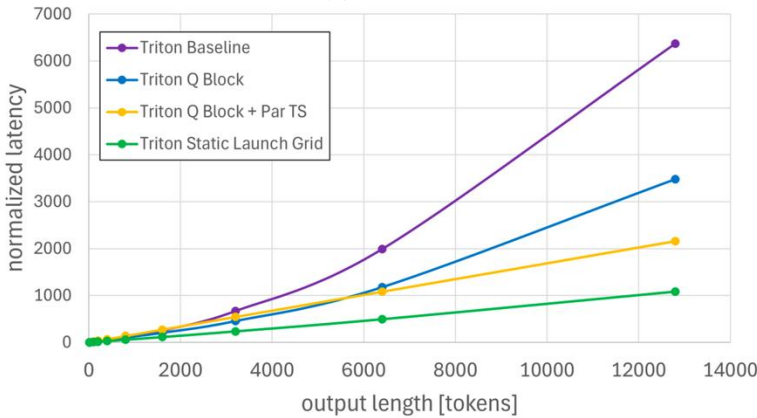
TABLE I
INVESTIGATED LLM KERNEL IMPLEMENTATIONS

	Implementation	LoC	Target vendor	Source
Attention	flash_attn	69197	NVIDIA	github.com/Dao-AI/flash-attention, [1], [10]
	rocm_flash_attn	52489	AMD	github.com/ROCm/flash-attention
	pytorch native	29	NVIDIA / AMD	pytorch/.../functional.py, [6]
	Triton manual	1049	NVIDIA / AMD	[11]
	Triton w/ autotuning	1100	NVIDIA / AMD	ibm.biz/vllm-ibm-triton-lib (this work)
RMS norm	layernorm_kernels.cu	159	NVIDIA (& AMD via hipify)	github.com/vllm-project/vllm, [3]
	Triton w/ autotuning	96	AMD / NVIDIA	ibm.biz/vllm-ibm-triton-lib (this work)

The Anatomy of a Triton Attention Kernel



(a) H100



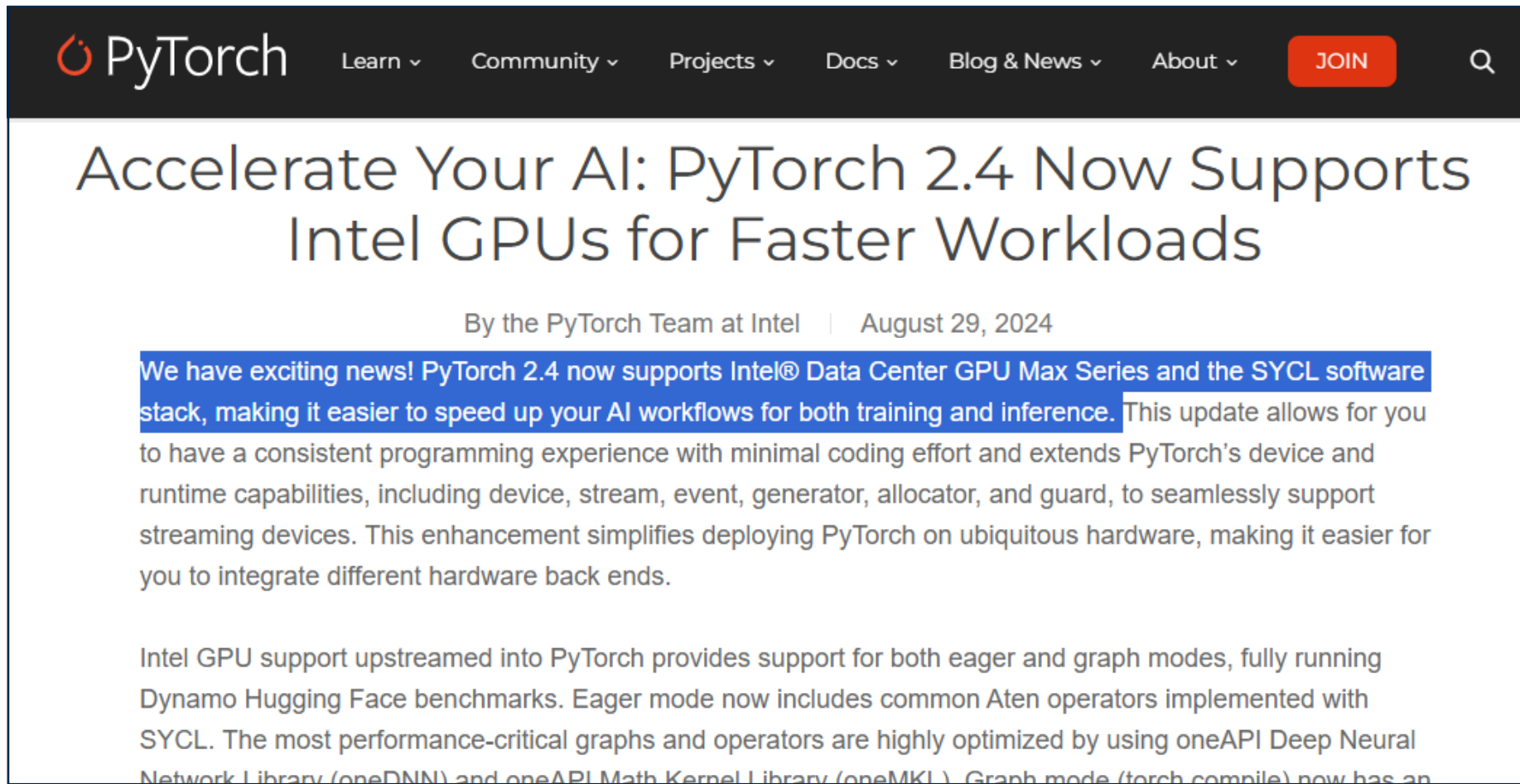
(b) MI300

Figure 9. vLLM latency benchmark results for Llama-3.1-8B with batch size 1 and input length of 500 tokens.

Sources: [arXiv: 2505.03780](#), [arXiv: 2511.11581](#) [vLLM blog - Triton article](#)

Triton on Intel GPUs, Shipping Since PyTorch 2.4

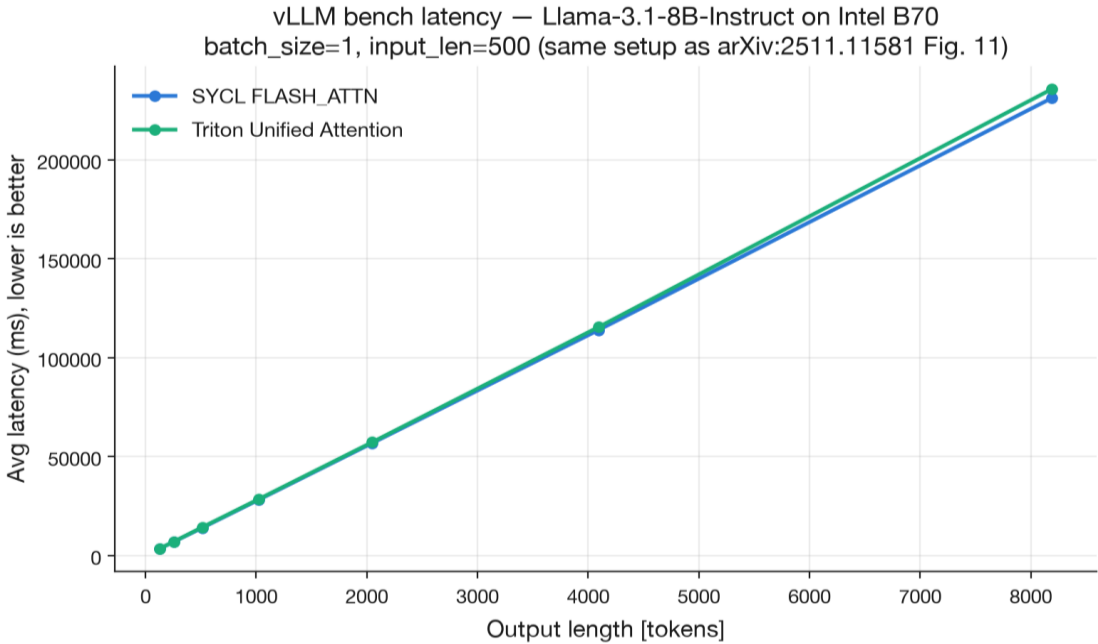
July 2024 — two years of releases, installed by default



The image is a screenshot of the PyTorch website's announcement page. The header features the PyTorch logo and navigation links: Learn, Community, Projects, Docs, Blog & News, and About, along with a red JOIN button and a search icon. The main heading reads "Accelerate Your AI: PyTorch 2.4 Now Supports Intel GPUs for Faster Workloads". Below this, it states "By the PyTorch Team at Intel | August 29, 2024". The first paragraph, highlighted with a blue background, says: "We have exciting news! PyTorch 2.4 now supports Intel® Data Center GPU Max Series and the SYCL software stack, making it easier to speed up your AI workflows for both training and inference." The rest of the paragraph continues: "This update allows for you to have a consistent programming experience with minimal coding effort and extends PyTorch's device and runtime capabilities, including device, stream, event, generator, allocator, and guard, to seamlessly support streaming devices. This enhancement simplifies deploying PyTorch on ubiquitous hardware, making it easier for you to integrate different hardware back ends." The second paragraph states: "Intel GPU support upstreamed into PyTorch provides support for both eager and graph modes, fully running Dynamo Hugging Face benchmarks. Eager mode now includes common Aten operators implemented with SYCL. The most performance-critical graphs and operators are highly optimized by using oneAPI Deep Neural Network Library (oneDNN) and oneAPI Math Kernel Library (oneMKL). Graph mode (torch.compile) now has an

One Kernel, Three Vendors

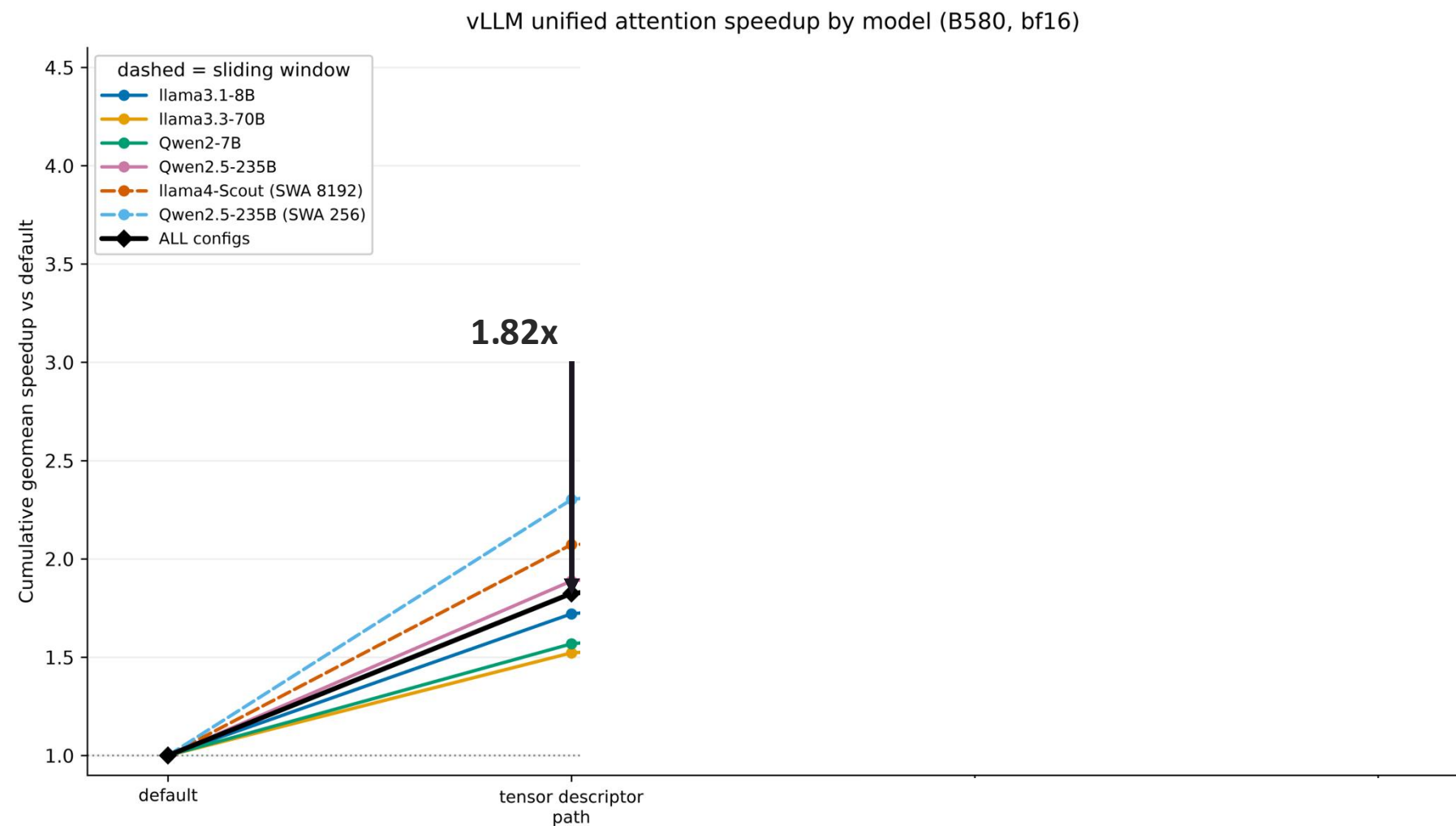
Immediate Functional Portability on Intel GPUs
— from the same Triton code as NVIDIA and AMD



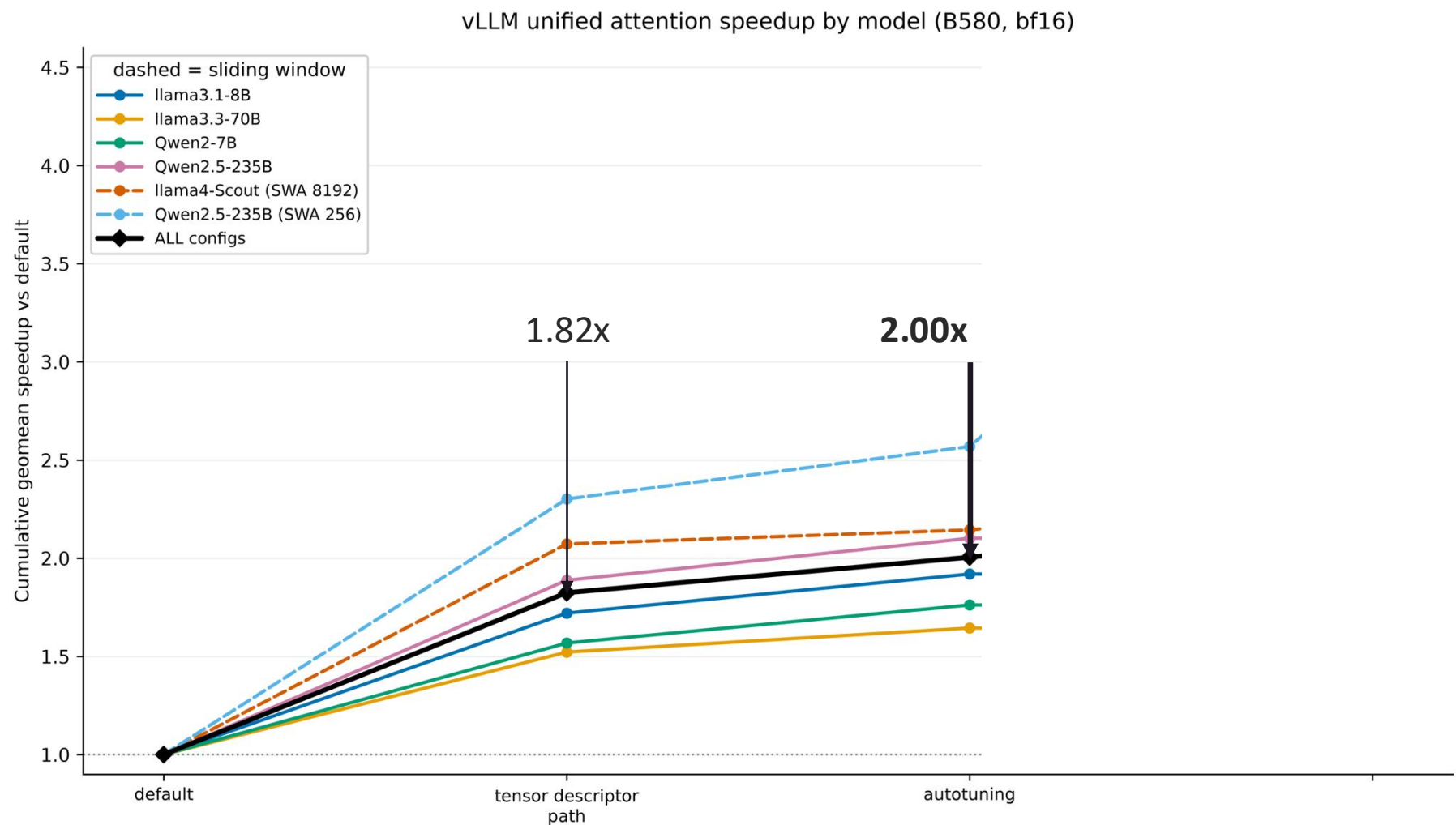
5.8x less of code

Implementation	LoC
SYCL FLASH_ATTN	6,877
Triton Unified Attention	1,194

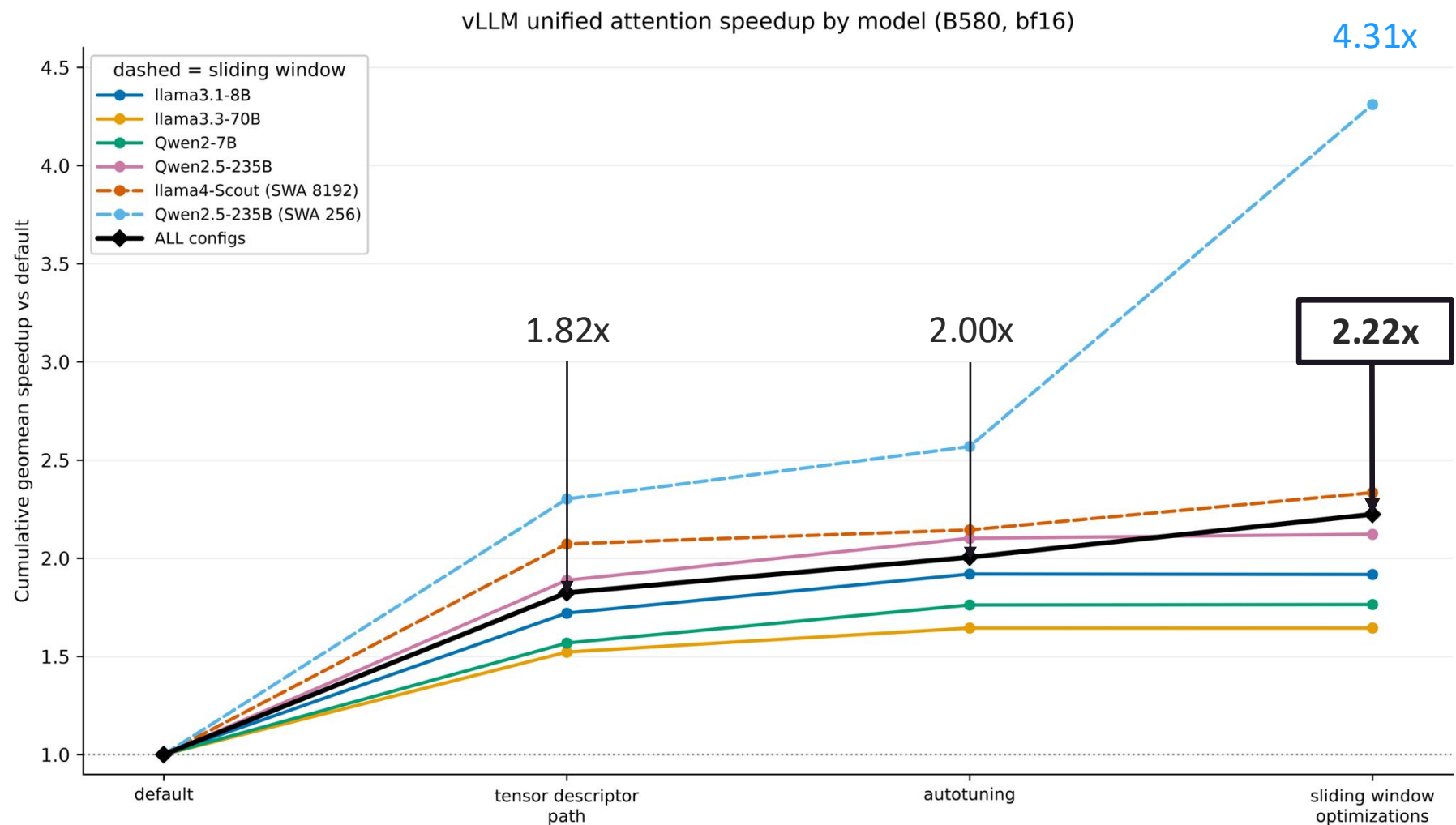
Case Study: Unified Attention



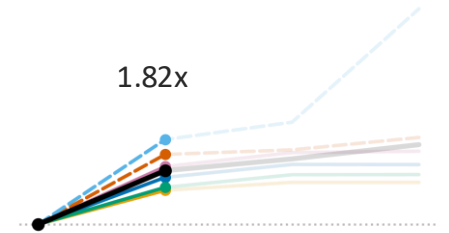
Case Study: Unified Attention



Case Study: Unified Attention



Deep Dive #1: Tensor Descriptor



[vLLM # attention: add USE TD constexpr for tensor descriptor Q/K/V load/store - #40327](#)

Tensor of Pointers

- Low-level memory access requiring manual address arithmetic, similar to CUDA or C
- Fine-grained control over memory access
- Flexibility for irregular access patterns

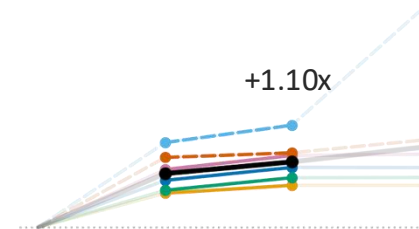
```
offs_m = tl.arange(0, BLOCK_M) % M
offs_n = tl.arange(0, BLOCK_N) % N
x_ptrs = x_ptr
          + (offs_m[:, None] * stride_M
            + offs_n[None, :] * stride_N)
mask = (offs_m < M)[:, None]
      & (offs_n < N)[None, :]
x = tl.load(x_ptrs, mask=mask, other=0.0)
```

Tensor Descriptor

- Abstract memory access using shape and stride metadata, similar to PyTorch tensors
- Clean, readable multidimensional tensors indexing
- Maps directly to Block IO on Intel and TMA on Nvidia
- Perform offset calculations and masking internally

```
x_desc = tl.make_tensor_descriptor(
    x_ptr,
    shape=[M, N],
    strides=[stride_M, stride_N],
    block_shape=[BLOCK_M, BLOCK_N])
x = x_desc.load([0, 0])
```

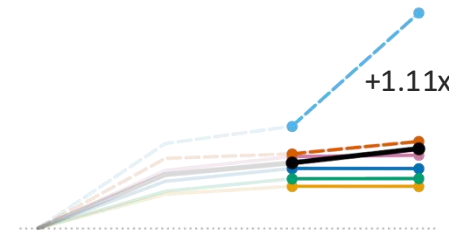
Deep Dive #2: Hardware-Aware Autotuning



- Simple parameter autotuning can deliver significant speedup over hardcoded values.
- Autotuning keys must be chosen carefully; broad enough to maximize re-use of tuned configurations and specific enough to separate workloads that need different configurations.
- Autotuning can be performed ahead-of-time and the best tuned configurations for the common workloads and hardware can be saved to avoid the cost of autotuning at runtime.

```
@triton.autotune(  
    configs=[  
        triton.Config(  
            {"BLOCK_M": block_m, "TILE_SIZE": tile_size},  
            num_stages=2,  
        )  
        for block_m in (16, 32, 64)  
        for tile_size in (16, 32)  
    ],  
    key=[  
        "BLOCK_SIZE",  
        "HEAD_SIZE",  
        "SLIDING_WINDOW",  
        ...  
    ],  
)  
@triton.jit  
def kernel_unified_attention(...):
```

Deep Dive #3: Sliding Window



- Defensively zeros out-of-window V before P@V
 - score masking guarantees that out-of-window P=0 $\rightarrow$ 0·V
 - zeroing is only necessary when V can be NaN/inf
- CUDA: uses custom_all_reduce, which leaks NaNs into the KV cache through the full graph capture warmup
- XPU: does not use custom_all_reduce $\rightarrow$ no NaN source $\rightarrow$ can remove redundant zeroing

```
if SLIDING_WINDOW:  
    qpos_lo = q_block_local_idx * BLOCK_Q  
    V = tl.where((context_len + qpos_lo - seq_offset[:, None]) < SLIDING_WINDOW, V, 0.0)
```

- Small sliding window workloads benefit from the 2D path

Summarization – updated kernels in vLLM

Kernel / Change	PR	Performance change
Unified attention (TD)	vllm#40327	+34-210%
Follow-up unified attention	vllm#43597	+1-10%

Source: [\[RFC\]: Tensor descriptor \(TD\) adoption strategy for vLLM Triton kernels vllm#42545](#)

Summarization – updated kernels in vLLM

Kernel / Change	PR	Performance change
Unified attention (TD)	vllm#40327	+34-210%
Follow-up unified attention	vllm#43597	+1-10%
Int4 MoE	vllm#52652	+5-12%
Mamba2 SSD (TD)	vllm#60801	+6-15%
Unquantized fused_moe_kernel (TD)	vllm#42436	+39%
Batched MoE tile resize / grid	vllm#51605	+66%
Batched MoE expert GEMM (TD)	vllm#46340	+116%
W8A8 int8 scaled_mm (TD)	vllm#47205	+139%

Source: [\[RFC\]: Tensor descriptor \(TD\) adoption strategy for vLLM Triton kernels vllm#42545](#)

Summarization – updated kernels in vLLM

Kernel / Change	PR	Performance change
Unified attention (TD)	vllm#40327	+34-210%
Follow-up unified attention	vllm#43597	+1-10%
Int4 MoE	vllm#52652	+5-12%
Mamba2 SSD (TD)	vllm#60801	+6-15%
Unquantized fused_moe_kernel (TD)	vllm#42436	+39%
Batched MoE tile resize / grid	vllm#51605	+66%
Batched MoE expert GEMM (TD)	vllm#46340	+116%
W8A8 int8 scaled_mm (TD)	vllm#47205	+139%

Source: [\[RFC\]: Tensor descriptor \(TD\) adoption strategy for vLLM Triton kernels vllm#42545](#)

Special Thanks to **Olena** and **Quinn**

Thank you for your early contributions and for helping establish the foundation of this work

PyTorch Conference 2026

Making vLLM Faster on Intel GPUs with Triton Kernels

Whitney Tsang, Principal Engineer, Intel
whitney.tsang@intel.com

Artur Fierka, AI Software Solution Engineer, Intel
artur.fierka@intel.com



Sources

[Triton's 2019 paper](#)



[OpenAI Triton's announcement](#)



[Semantic Scholar](#)



[arXiv: 2505.03780](#)



[arXiv: 2511.11581](#)



[vLLM blog - Triton article](#)



The Intel logo is centered on a solid blue background. It features the word "intel" in a white, lowercase, sans-serif font. A small, light blue square is positioned above the first vertical stroke of the letter 'i'. To the right of the word "intel" is a small white registered trademark symbol (®).

intel®