PyTorch CONFERENCE EUROPE 2026

# A CuTeDSL GEMM Backend for PyTorch Inductor

Nikhil Patel
Meta Superintelligence Labs

# Hardware is evolving faster than compiler GEMM kernels

**GEMMs remain the primary throughput bottleneck in transformer-based models**

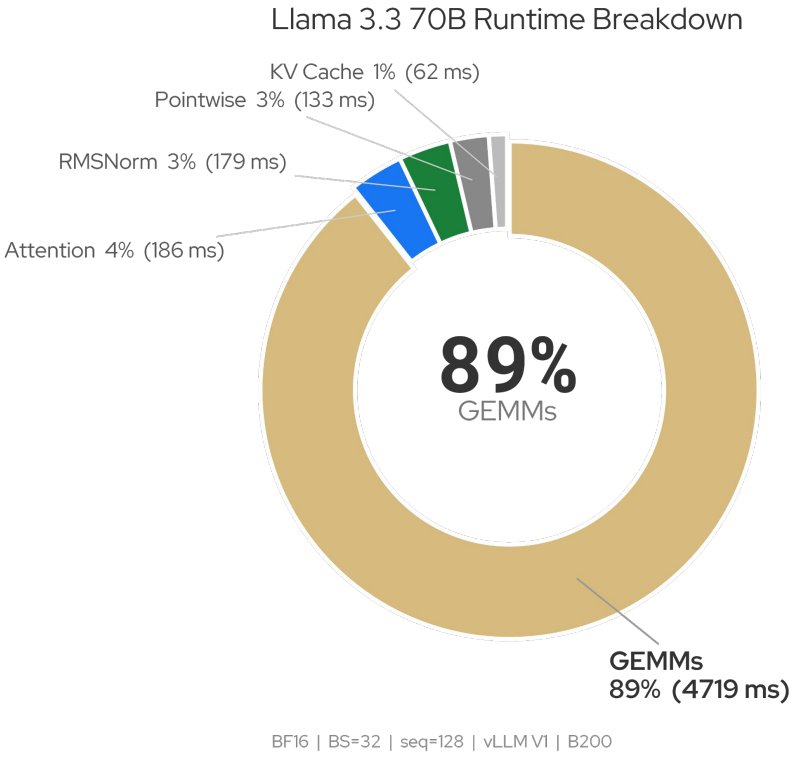
**Users expect `torch.compile` to deliver high-performance GEMMs**

- No custom kernels
- No manual tuning

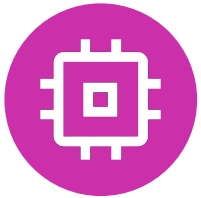
**On new architectures like Blackwell, that contract is breaking:**

Hardware ships new capabilities every generation:

- Tensor core modes
- Low-precision formats
- Scheduling features



# CuTeDSL: a scalable path to SOTA GEMMs



## Full thread and memory hierarchy exposure

Full use of architecture-specific features unlocks a high performance ceiling.

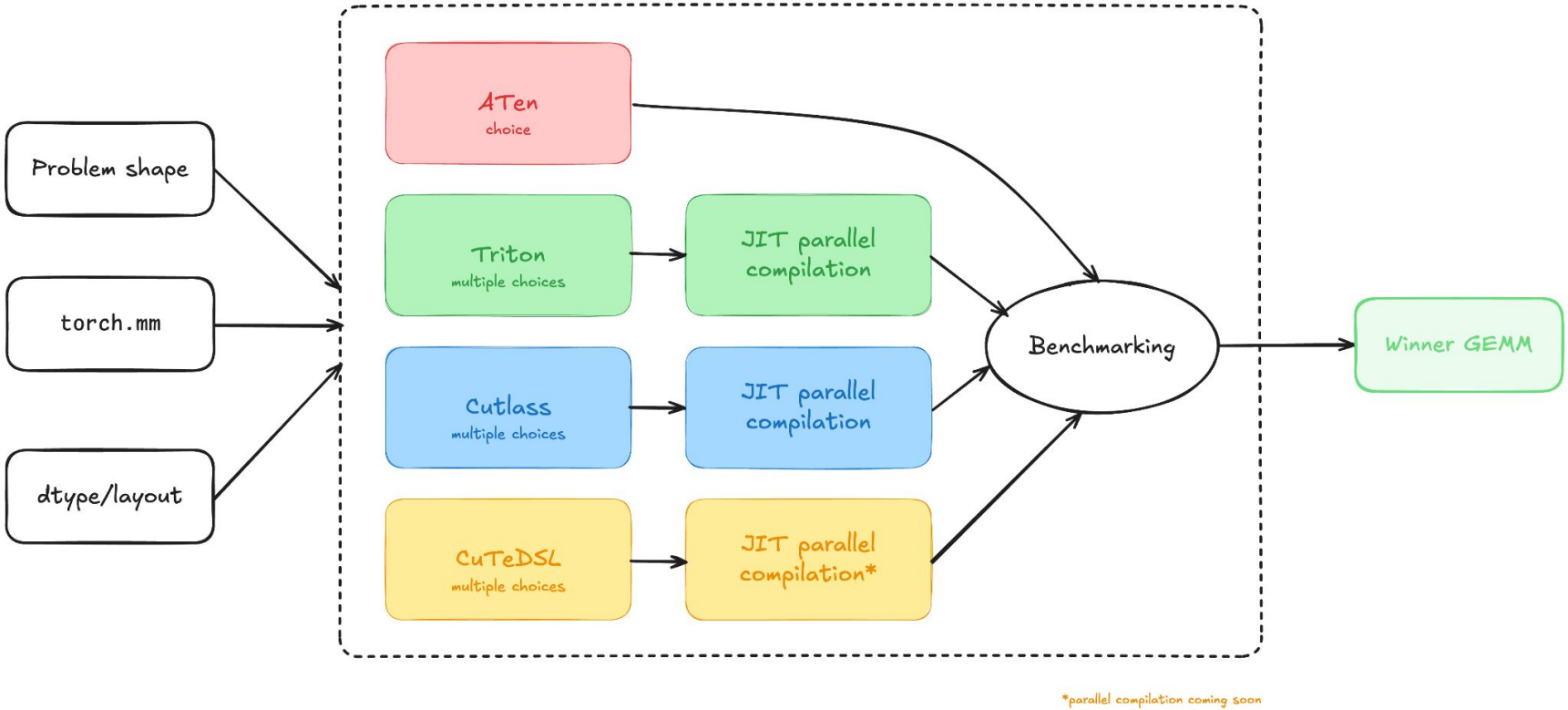
## Low compile time

Compile-time parity with Triton enables epilogue fusion benchmarking and autotuning across kernel configurations.

## Active NVIDIA investment

NVIDIA's early hardware access enables faster CuTeDSL support for new hardware capabilities.

# A CuTeDSL GEMM backend in PyTorch Inductor



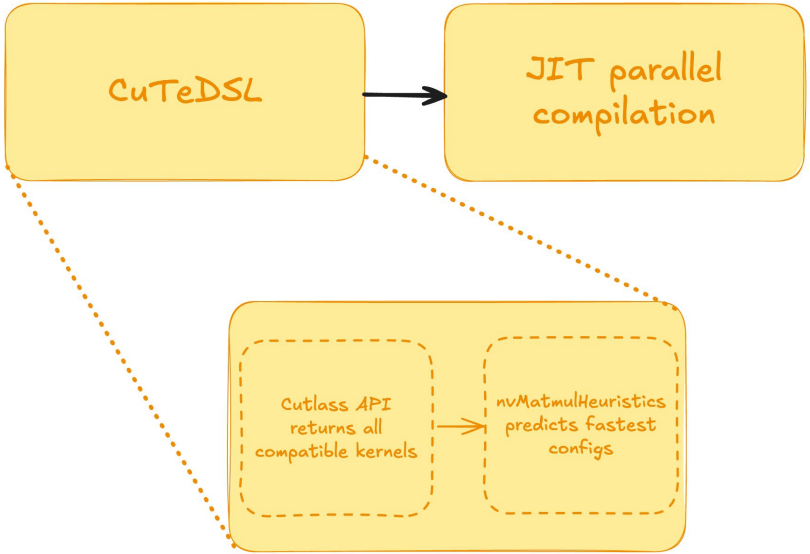
# The Cutlass API and nvMatmulHeuristics

## Cutlass API

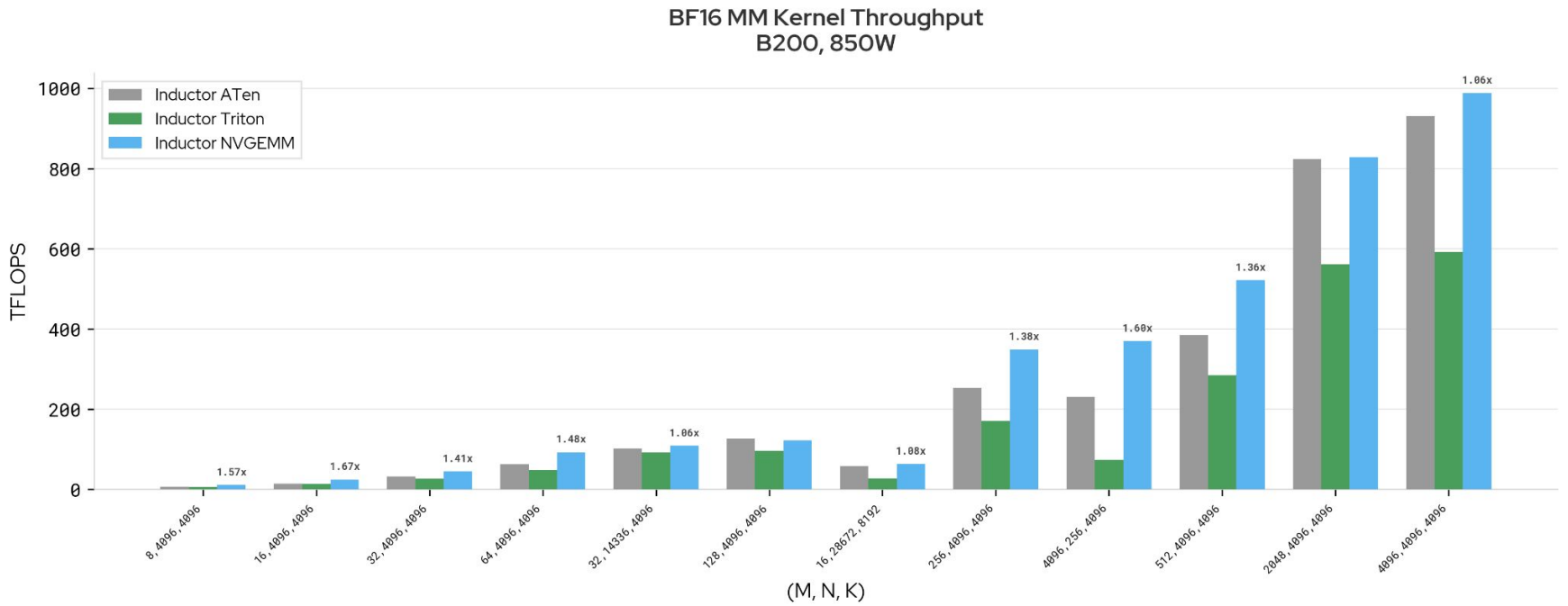
- NVIDIA-maintained kernel library
- Query (shape, dtype, layout, scaling) → all compatible kernels
- New hardware features and formats land without Inductor changes

## `nvMatmulHeuristics`

- Analytical model: scores configs by estimated hardware throughput
- Hundreds of candidates → top ~5 for on-device profiling
- Profiled alongside ATen and Triton — fastest wins

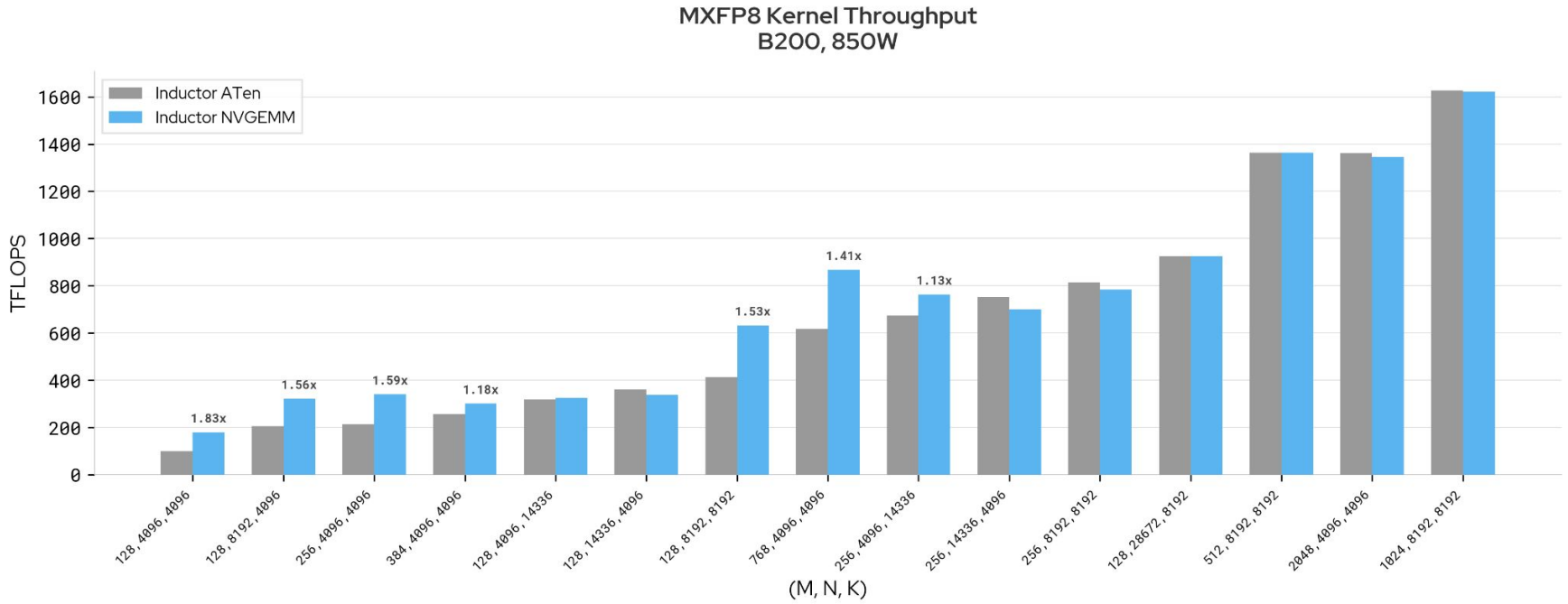


# BF16 Kernel Speedup



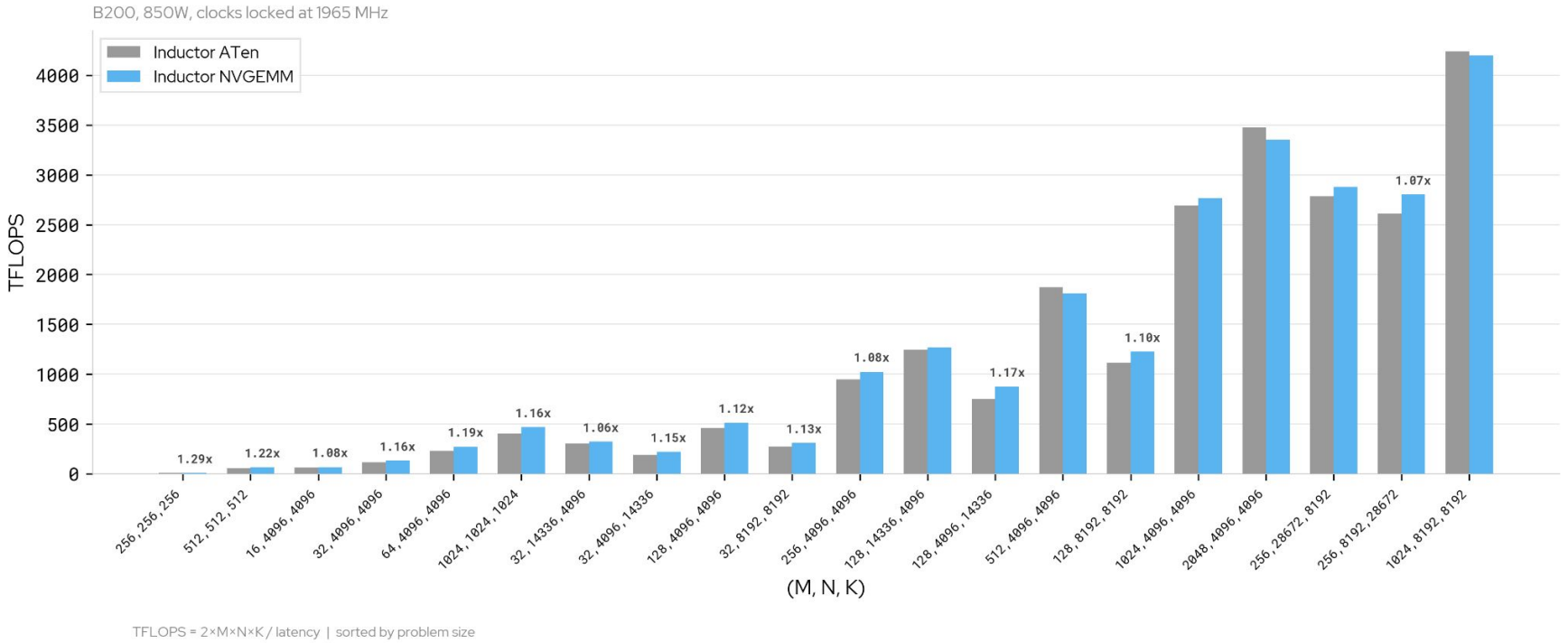

TFLOPS = 2×M×N×K / latency | sorted by problem size | callout = speedup vs best baseline

# MXFP8 Kernel Speedup



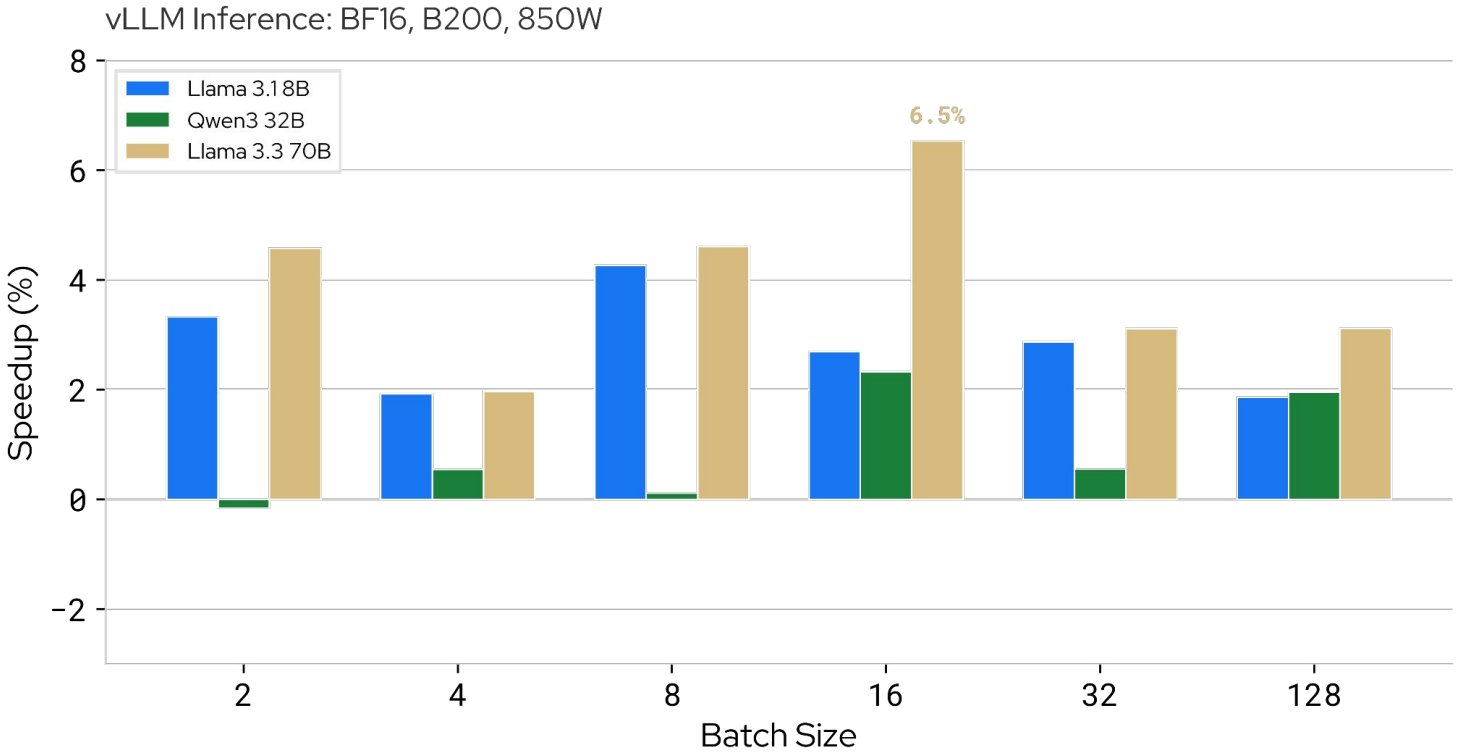

TFLOPS = 2×M×N×K / latency | sorted by problem size | callout = speedup vs ATen

# NVFP4 Kernel Speedup




TFLOPS = 2×M×N×K / latency | sorted by problem size

# End to End Model Speedup




Baseline: ATEN+TRITON | Treatment: ATEN+TRITON+NVGEMM | torch.compile, max-autotune

# Current Support

| PyTorch Op | A/B Dtypes | Output Dtypes | Scale Dtype | Block Size | Layout |
|---|---|---|---|---|---|
| **mm / bmm** | fp16, bf16, fp8, int8, uint8 | fp32, fp16, bf16, fp8, int32, int8, uint8 | – | – | any |
| **_scaled_mm** | | | | | |
| ↳ MXFP8 | fp8 (e4m3 / e5m2) | fp32, fp16, bf16, fp8 | e8m0fnu | 1×32 | any |
| ↳ MXF4 | fp4 (e2m1fn) | fp32, fp16, bf16, fp8 | e8m0fnu | 1×32 | TN only |
| ↳ NVF4 | fp4 (e2m1fn) | fp32, fp16, bf16, fp8 | e8m0fnu / e4m3fn | 1×16 | TN only |
| **_grouped_mm** | fp16, bf16, fp8 | fp32, fp16, bf16 | – | – | TN only |

**Requirements:** 16B-aligned · A=B dtype · max-autotune · fp32 acc · SM100+

**Coming Soon:** SM90 · Epilogue fusion · Block-scaled grouped GEMM · Grouped GEMM heuristics · AOT Compilation

# Usage

## Install required dependencies:

```
pip install nvidia-cutlass-dsl==4.3.5
pip install nvidia-matmul-heuristics

git clone --branch cutlass_api https://github.com/NVIDIA/cutlass.git
cd cutlass/python/cutlass_api
pip install -e ".[torch]"
```

## Add “NVGEMM” to the list of Inductor backends:

- Ops will automatically fall back to Triton or ATen if NVGEMM doesn’t support them

```
import torch._inductor.config as config
config.max_autotune_gemm_backends = "ATEN,TRITON,NVGEMM"
model = torch.compile(model, mode="max-autotune")  # no other changes
```

# Learn More

## PyTorch Blog Post:

pytorch.org/blog/gemms-torchinductor-cutedsl-backend/

## Source Code:

github.com/pytorch/pytorch/tree/main/torch/_inductor/codegen/nv_universal_gemm

CuTeDSL GEMM Backend – PyTorch Blog

# Q & A