TensorRT NVFP4 Profiling Report

Vision Transformer (ViT) Performance Analysis on NVIDIA Blackwell GPU

Date: December 6, 2025
GPU: NVIDIA RTX PRO 6000 Blackwell Workstation Edition
Model: ViT-Base (Batch Size 64, 224x224 input)

Executive Summary

This report analyzes the performance of NVFP4 (4-bit floating point) quantized Vision Transformer models on NVIDIA Blackwell GPUs using TensorRT. Key findings:

- **TensorRT 10.14 delivers 20% better NVFP4 performance** compared to TensorRT 10.11
- **NVFP4 achieves 1.62x speedup** over FP16 baseline (TRT 10.14)
- **Block-scaled FP4 GEMMs** are utilized on Blackwell tensor cores
- **Attention layers remain in FP16** due to weight-only quantization design

---

1. Test Configuration

1.1 Hardware and Software

Component	Specification
GPU	NVIDIA RTX PRO 6000 Blackwell (SM 12.0)
CUDA	12.9
Model	ViT-Base/16 (86M parameters)
Batch Size	64
Input Resolution	224 x 224 x 3
Quantization	NVFP4 (ModelOpt)

1.2 Container Images and TensorRT Versions

TensorRT 10.11 Setup:

Component	Value
Base Container	nvcr.io/nvidia/tensorrt:25.06-py3
TensorRT Version	10.11.0.33
Container OS	Ubuntu 24.04.2 LTS

TensorRT 10.14 Setup:

Component	Value
Base Container	nvcr.io/nvidia/tensorrt:25.06-py3

Component	Value
TensorRT Tarball	TensorRT-10.14.1.48.Linux.x86_64-gnu.cuda-12.9.tar.gz
TensorRT Version	10.14.1.48
Custom Image	tensorrt-10.14-blackwell:latest

The TensorRT 10.14 container was built by overlaying the official TensorRT 10.14 GA tarball onto the base container, replacing /opt/tensorrt with the newer version.

---

2. Performance Results

2.1 Latency Comparison

Model	TRT 10.11	TRT 10.14	Improvement
FP16 (Baseline)	8.30 ms	8.26 ms	-0.5%
NVFP4	6.12 ms	5.11 ms	**-16.5%**
NVFP4 vs FP16 Speedup	1.36x	**1.62x**	+19%

2.2 Throughput Comparison

Model	TRT 10.11	TRT 10.14	Improvement
FP16	120.5 qps	120.8 qps	+0.2%
NVFP4	162.9 qps	**195.5 qps**	**+20%**

2.3 Engine Size

Model	Engine Size	Compression
FP16	175 MB	1.0x (baseline)
NVFP4	56 MB	**3.1x smaller**

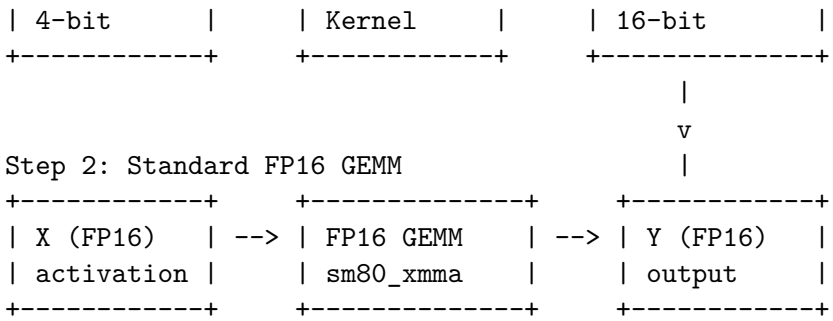
---

3. Key Technical Difference: TRT 10.11 vs 10.14

The most significant improvement in TensorRT 10.14 is HOW it executes NVFP4 linear layers. Both versions store weights in FP4 format, but they execute differently:

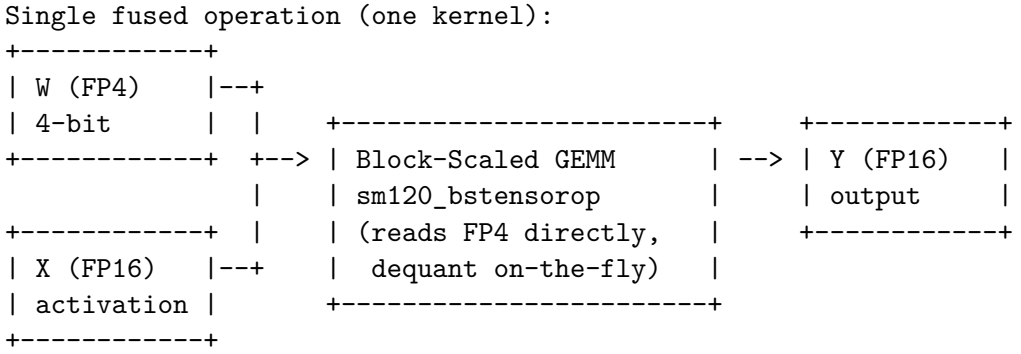
3.1 TRT 10.11: Two-Step Approach (Dequantize then FP16 GEMM)

```
Step 1: Dequantize weights (separate kernel)
+------------+     +------------+     +--------------+
| W (FP4)    | --> | Dequantize | --> | W (FP16)     |
```



Problems:
- Two kernel launches (overhead)
- FP16 weights need temporary memory buffer
- Extra memory bandwidth (read FP4, write FP16, read FP16)
- Uses older sm80 architecture kernels

3.2 TRT 10.14: Single Fused Block-Scaled GEMM



Benefits:
- Single kernel launch
- No intermediate FP16 buffer needed
- 4x less memory bandwidth for weights
- Uses Blackwell sm120 optimized kernels

3.3 Performance Impact of Kernel Difference

Aspect	TRT 10.11	TRT 10.14	Benefit
Kernel launches	2 (dequant + GEMM)	1 (fused)	Less overhead
Weight memory read	Read FP4, write FP16, read FP16	Read FP4 once	4x less bandwidth
Intermediate buffer	Required (FP16 weights)	None	Less memory
Architecture	sm80_xmma (older)	sm120_bstensorop (Blackwell)	Native support

3.4 Kernel Evidence from Profiling

TRT 10.11 NVFP4 kernel breakdown:

```
26.4%  sm80_xmma_gemm_f16f16_f16f16_f16  <-- Standard FP16 GEMM (dequantized)
24.3%  cutlass3x_sm120_block_scaled      <-- Some block-scaled
```

TRT 10.14 NVFP4 kernel breakdown:

```
 0.0%  sm80_xmma_gemm_f16f16             <-- ELIMINATED!
38.7%  cutlass3x_sm120_block_scaled      <-- ALL block-scaled FP4
```

TRT 10.14 completely eliminates the FP16 GEMM kernels for linear layers, replacing them with optimized block-scaled kernels that read FP4 weights directly.

---

4. Kernel-Level Analysis

4.1 FP16 Baseline Kernel Breakdown (TRT 10.14)

Kernel Type	% GPU Time	Description
FP16 GEMM (fused)	30.2%	Linear layers (MLP, projections)
FP16 GEMM (execute)	26.3%	Linear layers
FP16 GEMM (transpose)	20.0%	Linear layers (transposed)
FP16 GEMM (FP32 accum)	8.4%	High-precision accumulation
Multi-Head Attention	7.0%	Q@K, Softmax, A@V
LayerNorm (fused)	4.5%	Normalization layers
Patch Embed Conv	2.4%	Initial convolution
Total	**100%**	**8.26 ms**

4.2 NVFP4 Kernel Breakdown (TRT 10.14)

Kernel Type	% GPU Time	Description
Block-Scaled FP4 GEMM	**38.7%**	Quantized linear layers
Multi-Head Attention	12.6%	Q@K, Softmax, A@V (FP16)
GELU (fused)	11.9%	Activation function
LayerNorm (fused)	8.5%	Normalization layers
Reshape/Transpose	5.4%	Tensor operations
Patch Embed Conv	4.0%	Initial convolution (FP16)
Dynamic Quantization	~10%	Quant/dequant overhead
Other	~9%	Miscellaneous
Total	**100%**	**5.11 ms**

4.3 TRT 10.11 vs 10.14: NVFP4 Kernel Changes

Kernel Type	TRT 10.11	TRT 10.14	Change
FP16 GEMM	**26.4%**	**0%**	ELIMINATED
Block-Scaled FP4 GEMM	24.3%	**38.7%**	+59%
Attention (MHA)	9.6%	12.6%	Higher %
GELU	10.1%	11.9%	Similar
LayerNorm	7.0%	8.5%	Similar

---

5. Why Attention Remains in FP16

5.1 Weight-Only Quantization (W4A16)

NVFP4 uses weight-only quantization: only weights are FP4, activations remain FP16.

```
Linear Layer (CAN use FP4 kernel):
X (FP16 activation) @ W (FP4 weight) --> Y (FP16)
                      ^
                      Pre-quantized at export time
                      Stored in model file

Attention MatMul (CANNOT use FP4 kernel):
Q (FP16 activation) @ K^T (FP16 activation) --> Scores
^                      ^
Both computed at runtime - neither can be pre-quantized!
```

5.2 What Is Quantized vs Not Quantized

Layer Type	Quantized?	Reason
QKV Projection	YES (FP4)	Static weights
Output Projection	YES (FP4)	Static weights
MLP FC1	YES (FP4)	Static weights
MLP FC2	YES (FP4)	Static weights
Q @ K^T	NO (FP16)	Dynamic activations
Attention @ V	NO (FP16)	Dynamic activations
Softmax	NO (FP16)	Numerical stability
LayerNorm	NO (FP16)	Numerical stability

5.3 Why Not 4x Speedup?

Theoretical FP4 speedup is 4x, but achieved speedup is 1.62x due to:

Factor	Impact	Explanation
Weight-Only Quantization	Major	Only weights are FP4; activations remain FP16
Attention in FP16	Significant	Q@K and A@V MatMuls use FP16 activations

Factor	Impact	Explanation
Quantization Overhead	~10%	Dynamic quantization/dequantization
Non-Linear Ops	Minor	LayerNorm, GELU, Softmax in FP16

---

6. Block-Scaled FP4 GEMM Kernel Details

TensorRT 10.14 uses specialized CUTLASS 3.x kernels for Blackwell:

```
Kernel name: cutlass3x_sm120_bstensorop_s16864gemm_block_scaled_ue4m3xe2m1

Components:
- cutlass3x    : CUTLASS 3.x library
- sm120        : Blackwell SM 12.0 architecture
- bstensorop   : Block-scaled tensor operations
- block_scaled : Block-wise quantization aware
- ue4m3xe2m1   : Microscaled format (E4M3 x E2M1)
- f32_f16_f16  : FP32 accumulator, FP16 input/output
```

---

7. Recommendations

For Maximum Performance:

1. **Use TensorRT 10.14+** for NVFP4 models on Blackwell
2. **Expect 1.5-1.7x speedup** over FP16 (not 4x)
3. **Monitor engine size** - 3x memory savings is significant

For Further Optimization:

1. Consider **FP8 for attention** if accuracy permits
2. Use **TensorRT-LLM** for LLM workloads (native FP4 support)
3. Wait for **W4A4 quantization** support (both weights and activations in FP4)

---

8. Conclusion

TensorRT 10.14 significantly improves NVFP4 performance on Blackwell GPUs through:

1. **Fused Block-Scaled GEMM kernels** that read FP4 weights directly (vs dequantize-then-GEMM)
2. **Complete elimination of FP16 GEMMs** for linear layers
3. **Specialized Blackwell kernels** (sm120_bstensorop)
4. **Full graph fusion** via Myelin optimization

While the 1.62x speedup is below the theoretical 4x, it represents meaningful acceleration with 3x memory savings, making NVFP4 valuable for deployment scenarios.

Report generated by profiling_blackwell analysis pipeline