



# Optimize Generative AI Inference with Quantization in TensorRT-LLM and TensorRT

Asma Beevi KT, Senior Deep Learning Engineer  
Zhiyu Cheng, Tech Lead Manager

03/18/2024



## Agenda

1. Overview of quantization in TensorRT-LLM & TensorRT
2. Quantized LLM deployment with TensorRT-LLM
  - Workshop: CodeLlama 34B on a single 24GB GPU
3. E2E Stable-Diffusion-XL 8-bit Quantization with TensorRT
  - Workshop: SDXL 8-bit inference with demoDiffusion
4. Q & A

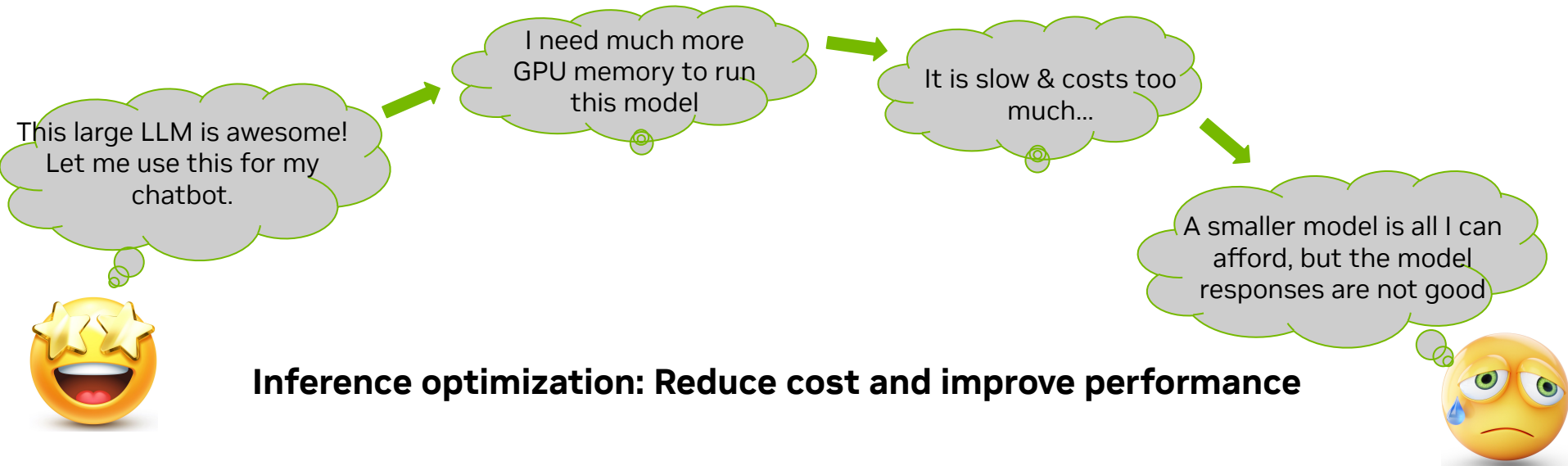


The background features a dark, abstract design. A diagonal line splits the image. The upper-left portion is dark with faint, concentric green circles. The lower-right portion shows a dense, glowing green pattern of horizontal lines and dots, resembling a digital or data visualization. 

# Overview of Quantization in TensorRT-LLM & TensorRT

# Generative AI Inference

Generative models are expensive to train, but their inference, a recurring cost is much more expensive!



**Inference optimization: Reduce cost and improve performance**

# Accelerated & Optimized Inference Solutions from NVIDIA

## TensorRT:

- Compiler and runtime optimized for accelerated deep learning inference on NVIDIA GPUs.
- Optimization techniques such as quantization, layer and tensor fusion, kernel tuning, etc.
- Diffusion models deployed via TensorRT

## TensorRT-LLM:

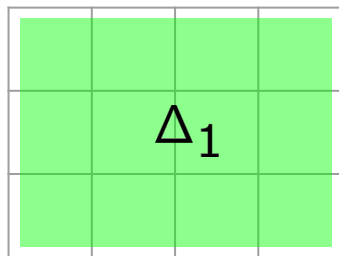
- Inference accelerator for Large Language Models (LLMs) on NVIDIA GPUs wrapping TensorRT
- Features fast transformer kernels, pre/post processing & multi-GPU/multi-node support
- Further speed up inference via quantization, KV cache, inflight batching etc
- LLM models deployed via TensorRT-LLM

# What is Quantization?

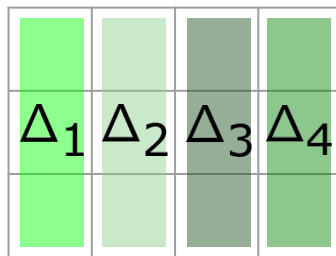
- Quantization is the process of converting tensors to low precision data types.
- Quantization reduces model size and increases computational speed.

Quantization:  $X_q = \lfloor \frac{X}{\Delta} \rfloor$ ,  $\Delta$  - Quantization scaling factor

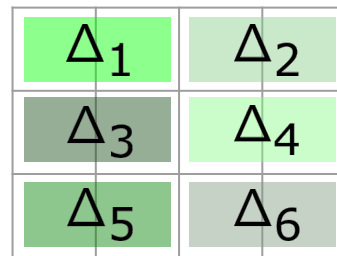
## Different Quantization Formats



Per Tensor Quantization

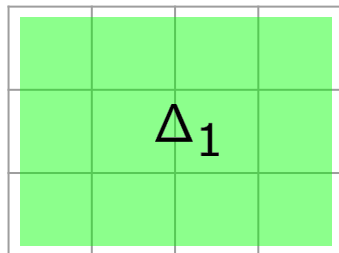


Per Channel Quantization

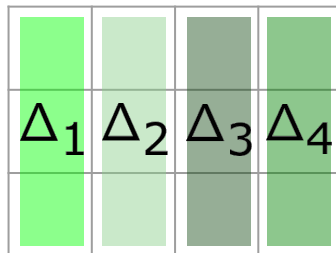


Per Block Quantization

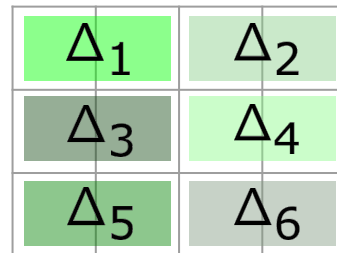
# What is Quantization?



Per Tensor Quantization



Per Channel Quantization



Per Block Quantization



# Taming the beast - Falcon 180B on a single GPU

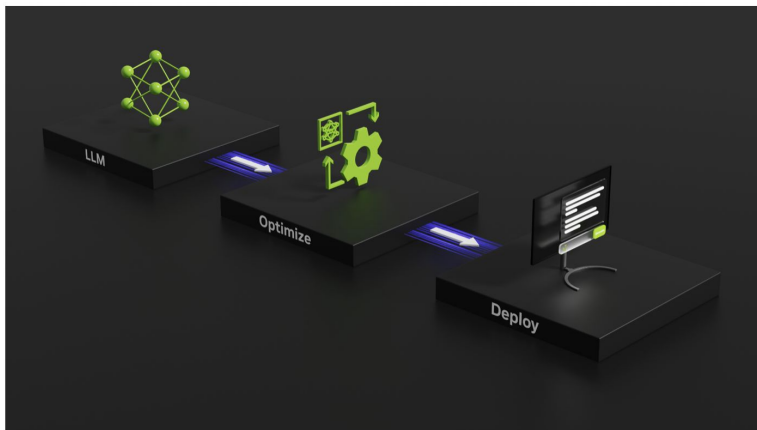
Quantization: enablement + accelerate

## NVIDIA TensorRT-LLM Enhancements Deliver Massive Large Language Model Speedups on NVIDIA H200

Dec 04, 2023

+12 Like Discuss (0)

By [Ashraf Eassa](#), [Dave Salvator](#) and [Nick Comly](#)



*“Falcon-180B is one of the largest and most accurate open-source large language models available, and previously required a minimum of eight NVIDIA A100 Tensor Core GPUs to run it.*

*...TensorRT-LLM advancements in a **custom INT4 AWQ** make it possible to run entirely on a single H200 Tensor Core GPU ...”*

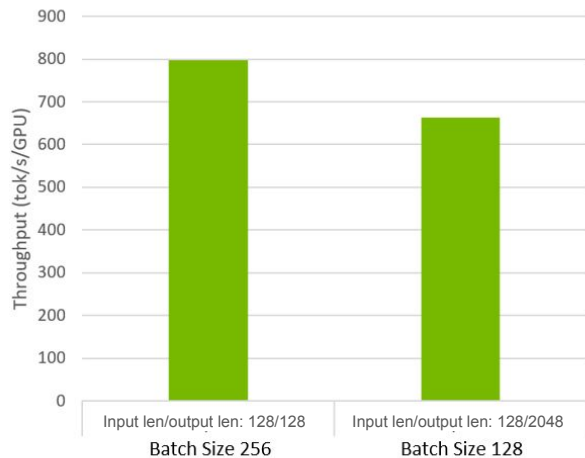
[Blog post](#)



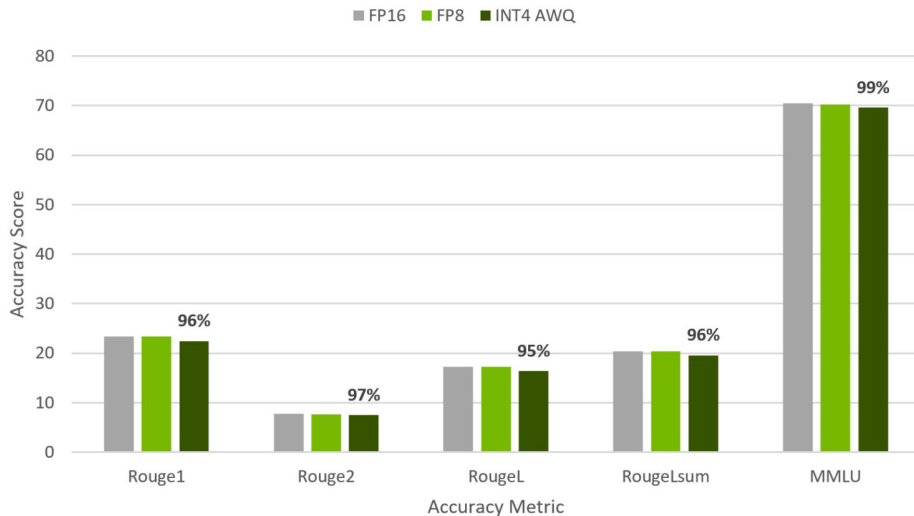
# Taming the beast - Falcon 180B on a single H200 GPU

Quantization: enablement + accelerate

Falcon-180B Single H200 Performance



Falcon-180B Accuracy



- Quantization is a very effective model optimization technique especially for large models like LLMs
- Quantization from TensorRT/TensorRT-LLM can compress model size by 2x - 4x, speeding up inference while retaining model quality.

## TensorRT/TensorRT-LLM Quantization Toolkit

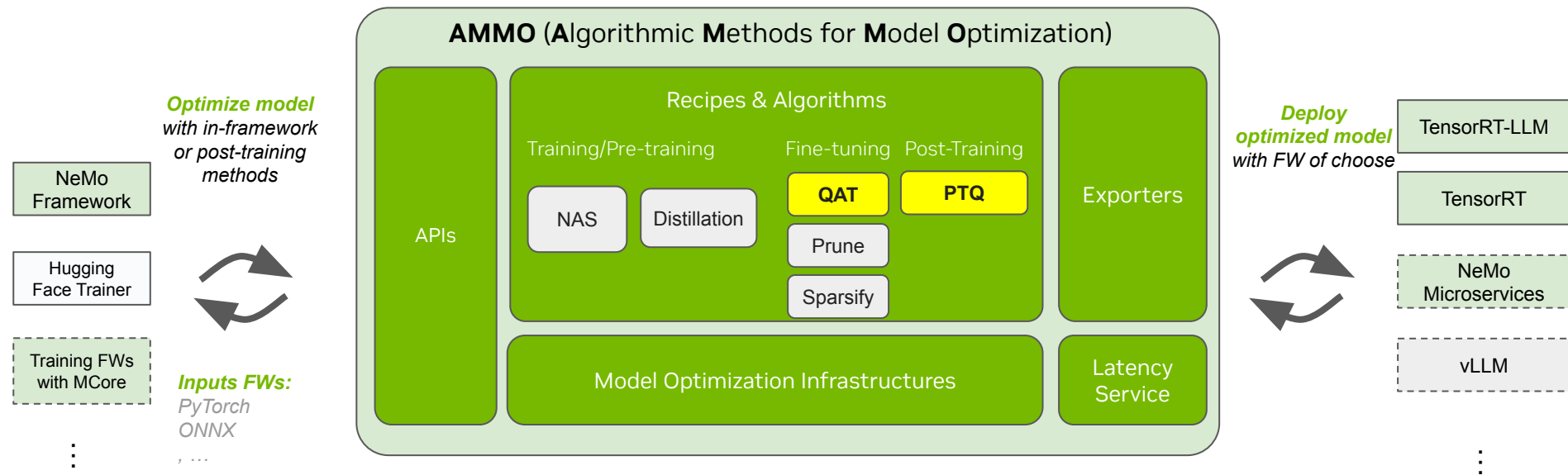
- Quantize and calibrate Pytorch models via **Post Training Quantization (PTQ)** and **Quantization Aware Training (QAT)**
- High performant quantization formats - FP8, INT8, INT4 etc
- Fast and accurate algorithms such as SmoothQuant, AWQ, Double Quantization with easy to use APIs
- Setups the model for deployment
  - Performs simulated quantization. Real speed up and compression come after deployment to TensorRT / TensorRT-LLM

Install via:

```
pip install nvidia-ammo --extra-index-url https://pypi.nvidia.com
```

# Bridging training frameworks to deployment

High-level view



Nvidia AI SW Stack

Coming soon

## TensorRT/TensorRT-LLM Quantization formats

Quantization format	Details	Deployment
FP8	<ul style="list-style-type: none"><li>Per-tensor FP8 weights and activations</li></ul>	<ul style="list-style-type: none"><li>TensorRT, TensorRT-LLM</li><li>GPUs: Ada, Hopper &amp; later</li></ul>
INT8	<ul style="list-style-type: none"><li>Per-channel INT8 weights, per-tensor INT8 activations</li><li>Default PTQ algorithm: <a href="#">SmoothQuant</a></li></ul>	<ul style="list-style-type: none"><li>TensorRT, TensorRT-LLM</li><li>Supported on most GPUs.</li></ul>
W4A16 (INT4 weights only)	<ul style="list-style-type: none"><li>Blockwise INT4 weights, FP16 activations</li><li>Default PTQ algorithm: <a href="#">AWQ</a></li></ul>	<ul style="list-style-type: none"><li>TensorRT-LLM</li><li>GPUs: Ampere &amp; later</li></ul>
W4A8 (INT4 weights, FP8 activations)	<ul style="list-style-type: none"><li>Blockwise INT4 weights, FP8 per-tensor activations</li><li>Default PTQ algorithm: <a href="#">AWQ</a></li></ul>	<ul style="list-style-type: none"><li>TensorRT-LLM</li><li>GPUs: Ada, Hopper &amp; later</li></ul>

## Fast & accurate PTQ : Example workflow

```
import ammo.torch.quantization as atq

model = ... # Load model

# Load calibration data-loader, a small subset of data is sufficient
calib_dataloader = get_calib_dataloader(num_samples=512)

# Define forward loop for calibration with the model as input
def forward_loop(model):
    for data in calib_dataloader:
        model(data)

# Perform PTQ: Add quantizer nodes and perform calibration
model = atq.quantize(model, atq.INT4_AWQ_CFG, forward_loop)

# deploy the model to TensorRT-LLM or to TensorRT (via ONNX)
```

## Closing the accuracy gap with quantization-aware training (QAT)

- QAT can further close the accuracy gap after PTQ
- Quantization toolkit's `atq.quantize` API insert the quantizers and setup the model for QAT
  - Use the original training pipeline to perform the QAT training
- Recommended QAT recipe:
  - QAT for 1-10% duration after the original training/fine-tuning
  - Use small learning rates. A default option is the final learning rate of original training
  - SGD optimizer works as good as Adam
    - SGD has lower GPU memory requirements than Adam 😊



# Example: QAT Workflow with Hugging Face Trainer

PTQ workflow followed by training

```
from transformers import Trainer
```

```
import ammo.torch.opt as ato
import ammo.torch.quantization as atq
```

```
# [Not shown] load model, tokenizer, data loader etc
trainer = Trainer(model=model, tokenizer=tokenizer, args=training_args, **data_module)
```

```
# [Not shown] Get a calib dataloader without any distributed sampler
def forward_loop(model):
    for i, data in enumerate(calib_dataloader):
        model(data)
```

```
# Quantize the model; The model should be unwrapped from any distributed wrapper
# The model may be wrapped in a DataParallel or DistributedDataParallel after `atq.quantize`
model = atq.quantize(model, atq.INT8_DEFAULT_CFG, forward_loop)
```

```
# Save the ammo quantizer states
torch.save(ato.ammo_state(model), "ammo_quantizer_states.pt")
```

```
# To resume training from a checkpoint, skip `atq.quantize` step & directly load the ammo quantizer states
# ato.restore_from_ammo_state(model, torch.load("ammo_quantizer_states.pt"))
```

```
trainer.train()
trainer.save_state()
```

The background features a dark, textured surface with vibrant green digital patterns. On the right side, there is a prominent, curved, metallic-looking structure that resembles a stylized letter 'C' or a futuristic architectural element, composed of many small, rectangular segments. The overall aesthetic is high-tech and digital.

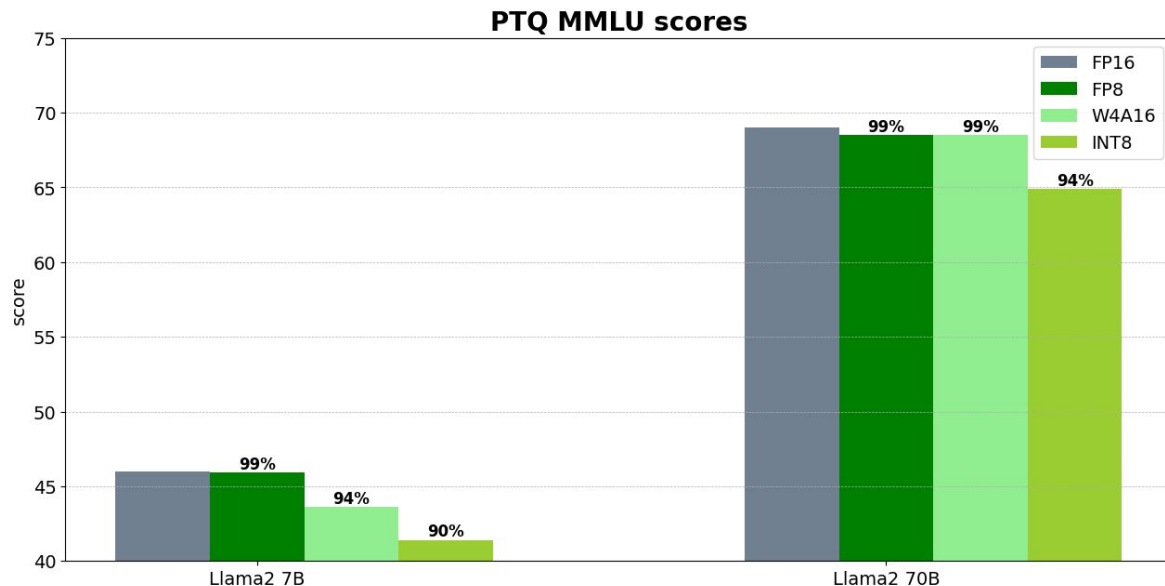
# Quantized LLM deployment with TensorRT-LLM

# Demo

Local Co-pilot: CodeLlama 34B Go Brrr on A SINGLE 24GB GPU

[Download link \(quantized codellama.ipynb\)](#)

# Selecting the right quantization: Accuracy



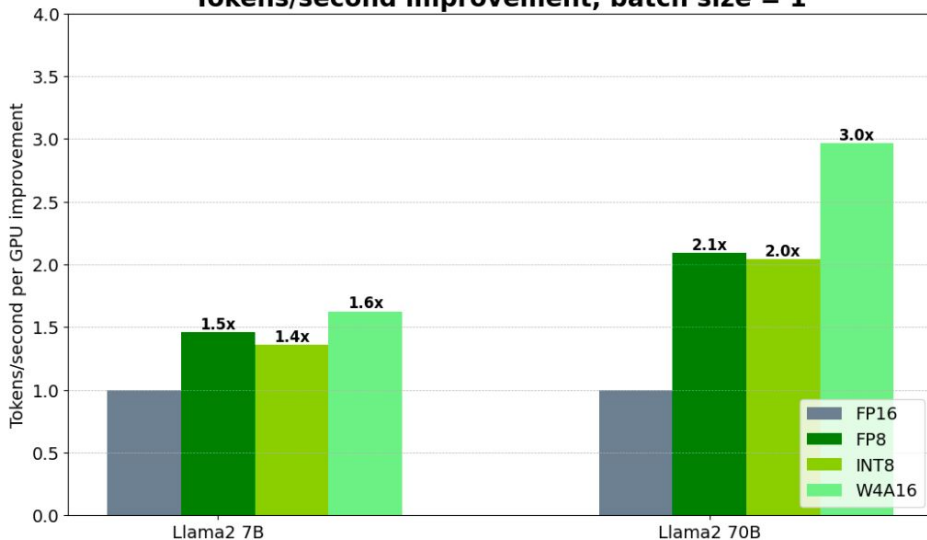
- FP8 quantization shows almost no accuracy degradation
- W4A16 (INT4 weight only) better than INT8 weight & activation quantization for model accuracy
  - Activations appear more sensitive to quantization than weights

# Selecting the right quantization: Memory-bound Vs Compute-bound

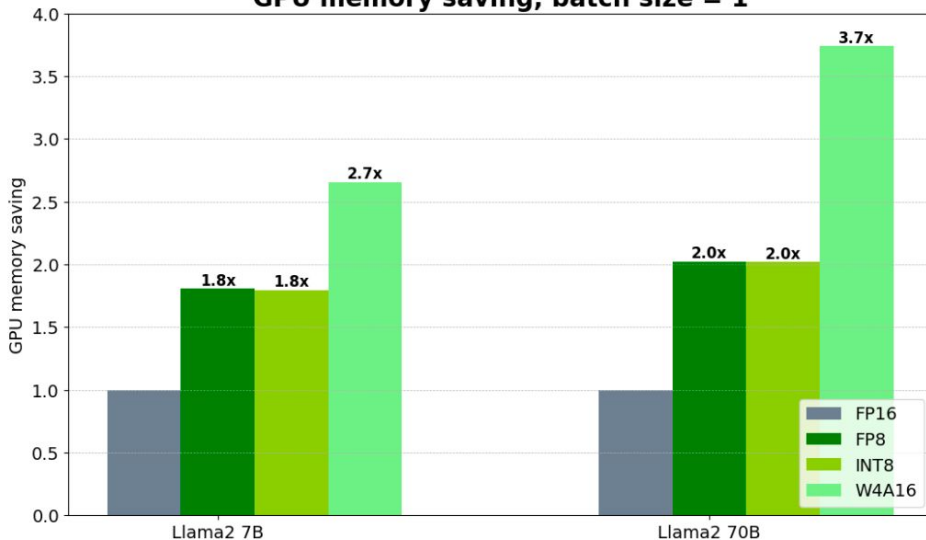
- **Memory-bound**
  - Inference is dominated by weight loading time from GPU memory to cache
  - Strategy: Quantize weights aggressively, may keep activations at higher precision to maintain quality
- **Compute-bound**
  - Inference is dominated by time taken by GPU for computations such as matrix multiplication
  - Strategy: Use lower precision computation kernels such as FP8/INT8 GEMM

## Selecting the right quantization: Small batch size

Tokens/second improvement, batch size = 1



GPU memory saving, batch size = 1

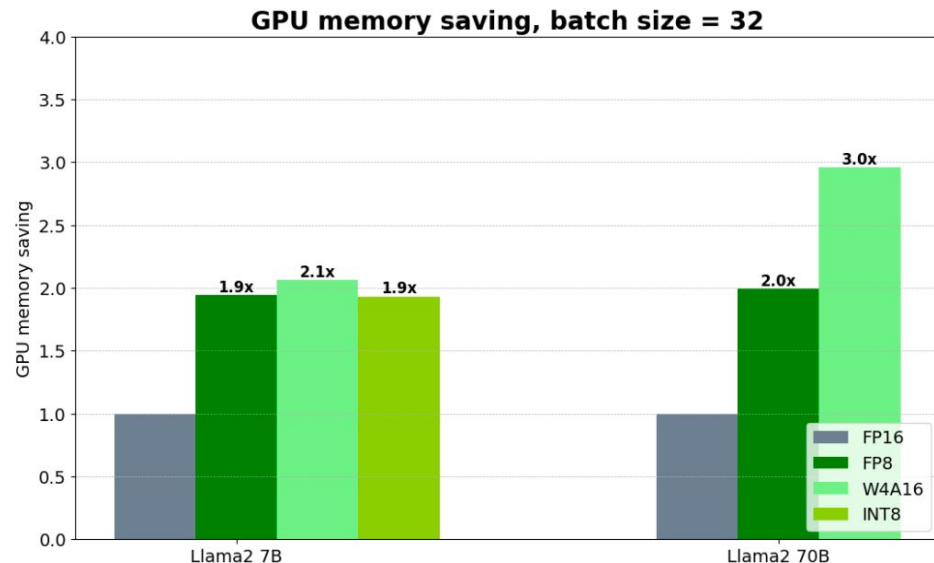
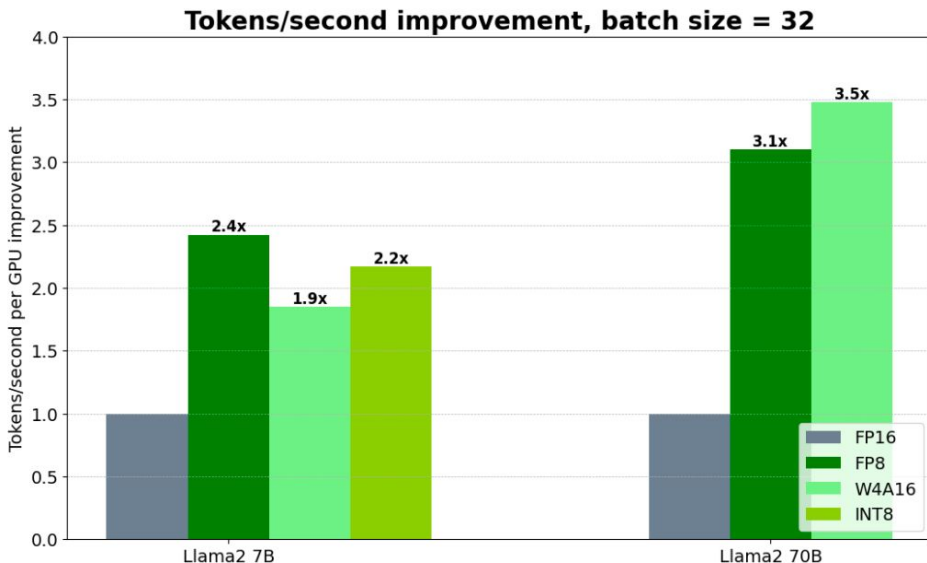


Device: H100, Input length: 2048, output length: 512  
H100 W4A16 kernels used in this benchmarking is available only from TRTLLM 0.9  
Quantized KV cache is used

- Small batch size generation is in **“Memory-bound”** regime
  - Aggressive weight quantization such as W4A16, W4A8 etc gives the most speed improvement
  - W4A16 uses FP16 kernel -> but still faster than FP8/INT8 using lower precision kernels



## Selecting the right quantization: Medium/large batch size



Device: H100, Input length: 2048, output length: 512  
H100 W4A16 kernels used in this benchmarking is available only from TRTLLM 0.9  
Quantized KV cache is used

- As batch size increases, generation becomes both **“Memory-bound”** and **“Compute-bound”**
  - Lower precision compute kernels such as FP8, INT8 or W4A8 more effective in compute-bound regime



# E2E Stable-Diffusion-XL 8-bit Quantization with TensorRT

Jingyu Xin, Zhiyu Cheng\*, Ajinkya Rasane, Chenjie Luo, Huizi Mao

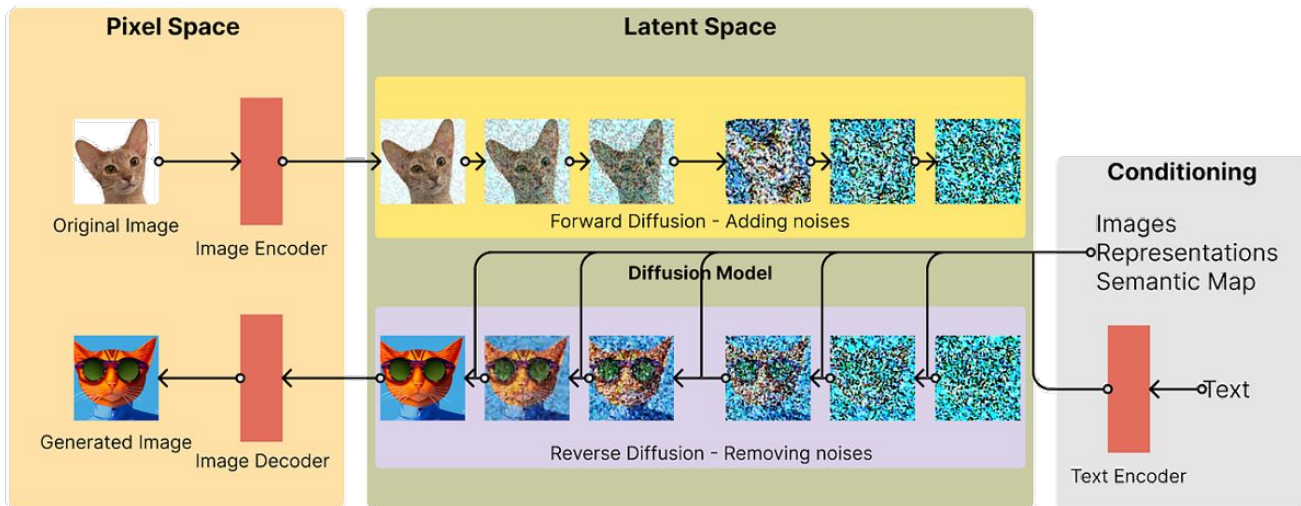
\*Presenter



## **Introduction: Stable Diffusion inference and quantization challenges**

# Stable Diffusion Introduction

- Stable diffusion is a diffusion based generative model that is widely used in applications such as text-2-image, text-to-video, text-to-3D, etc.
- Different from LLMs and CNNs, the fundamental idea of diffusion models is to view the generative process as an iterative procedure, where noise is gradually removed step-by-step from a random initial state.

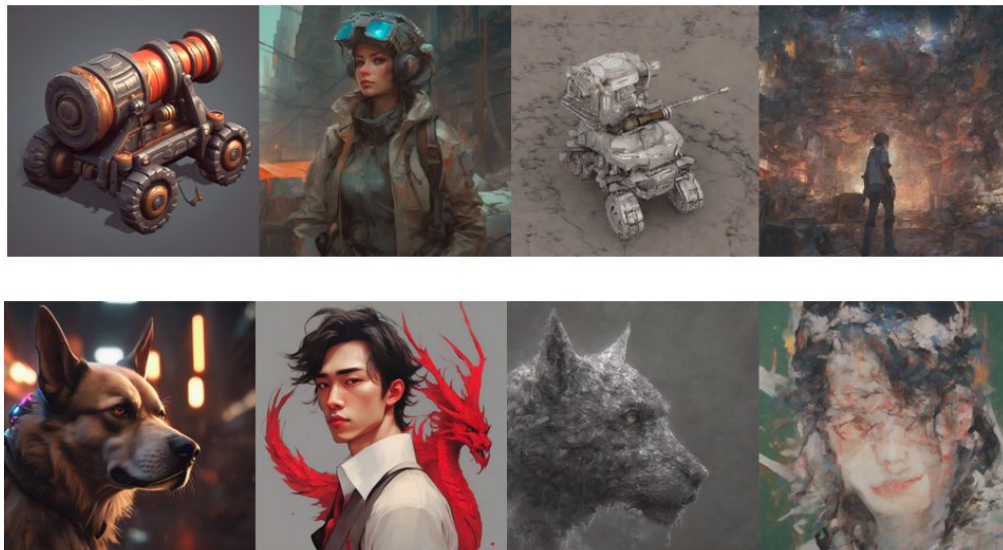


# Challenges and Optimizations for Stable Diffusion Inference

- Because of the inclusion of a sizable backbone model like UNet and a multi-step denoising process, conducting stable diffusion inference is both slow and resource-intensive, resulting in high memory usage and costs.
- Optimizing diffusion models to minimize inference latency, decrease model size, and lower costs is essential for enabling broader applications.
- Optimization techniques
  - Distillation
  - Efficient backbone model architecture
  - Efficient samplers
  - Fast kernels
  - Caching
  - **Quantization**



# Challenges in Stable Diffusion Quantization




FP16

Naïve INT8 quantization

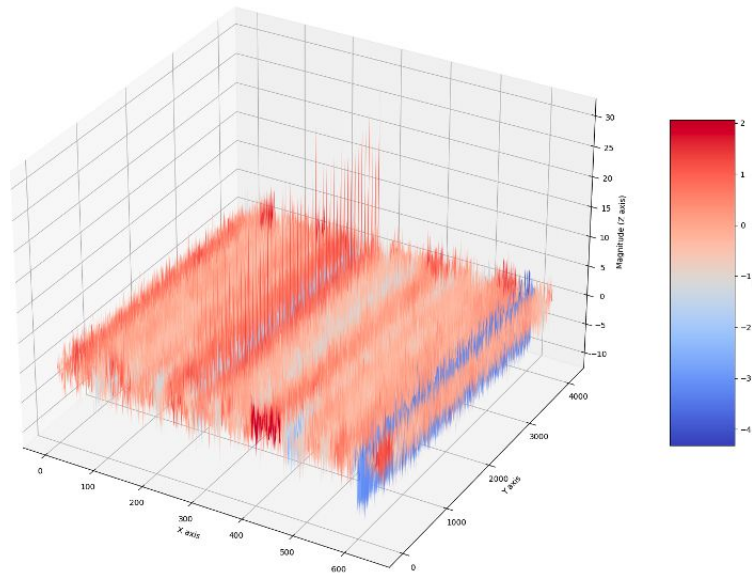
- Post-Training Quantization (PTQ) is commonly used for compression and acceleration in AI tasks but is not directly applicable to diffusion models due to their **unique multi-timestep denoising process**.
- The varying output distribution of the noise estimation network at each time step in diffusion models poses a challenge for PTQ calibration, rendering **a naive approach ineffective**.





**TensorRT's solution: accelerate inference  
with 8-bit quantization**

# A deeper look into stable diffusion quantization challenges

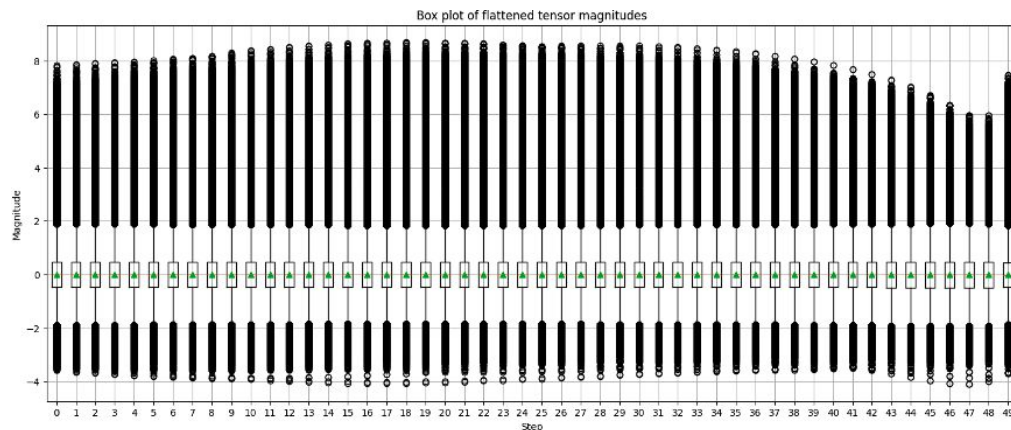


Activation distribution from different channels

Activation distributions in Stable Diffusion vary significantly

- Across different channels
- Varying noise levels

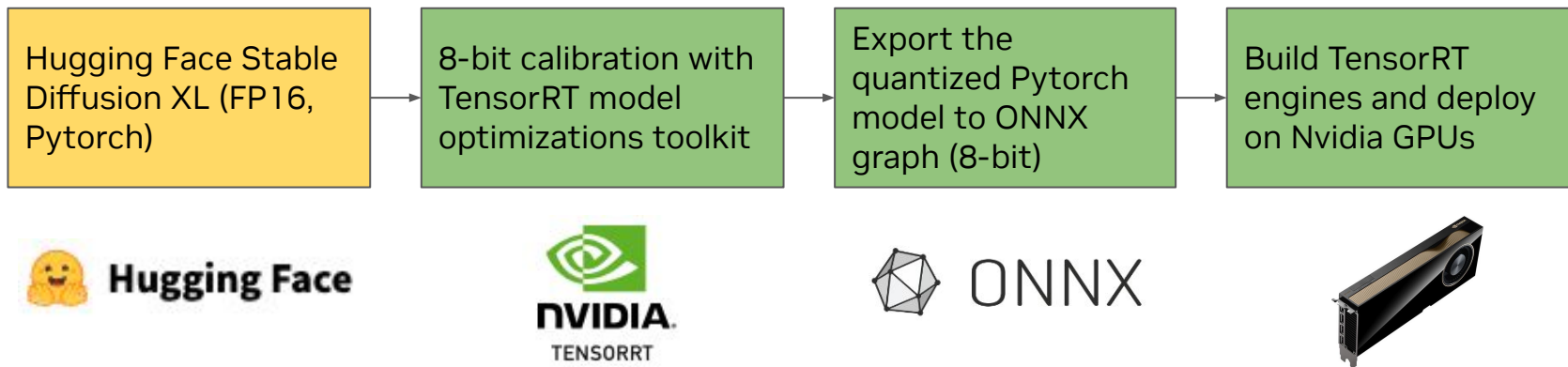
**Vanilla max quantization oversimplifies calibration based on the global max activation value.**



Activation distribution from different noise levels

## Stable Diffusion Quantization Pipeline with TensorRT

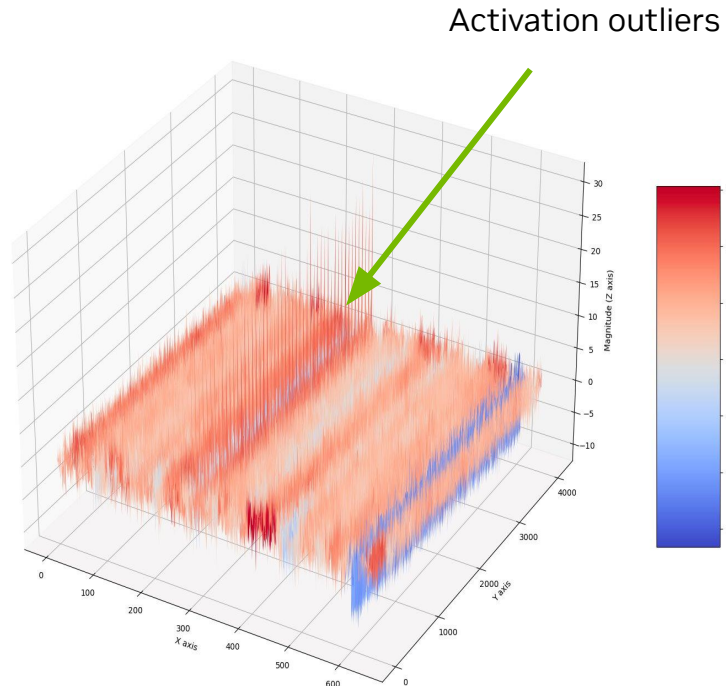
- To overcome the challenges in the post-training quantization for stable diffusion, NVIDIA TensorRT developed a sophisticated pipeline to facilitate low precision inference on NVIDIA GPUs, to achieve speedup, reduce memory usage, while maintain the generated image quality.



# Why TensorRT's quantization approach is unique

SmoothQuant, tailored to stable diffusion

- SmoothQuant [Xiao, et.al, ICML 2023]:
  - A popular PTQ method to enable 8-bit weight, 8-bit activation (W8A8) quantization for LLMs.
  - Key idea: Smooth the activation outliers by migrating the quantization difficulty from activations to weights with a mathematically equivalent transformation.
- TensorRT quantization toolkit includes a fine-grained tuning pipeline to determine the optimal parameter settings for each layer of the stable diffusion backbone model UNet for SmoothQuant.



Activation distribution from different channels

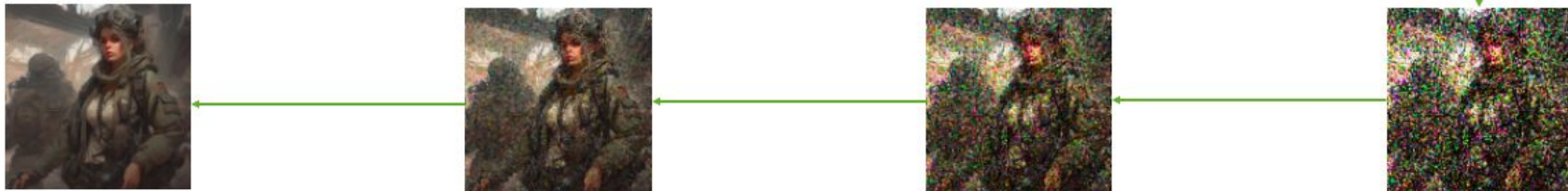
# Why TensorRT's quantization approach is unique

Determine quantization scaling factors in selected noise range

- Style, shape [High noise range]



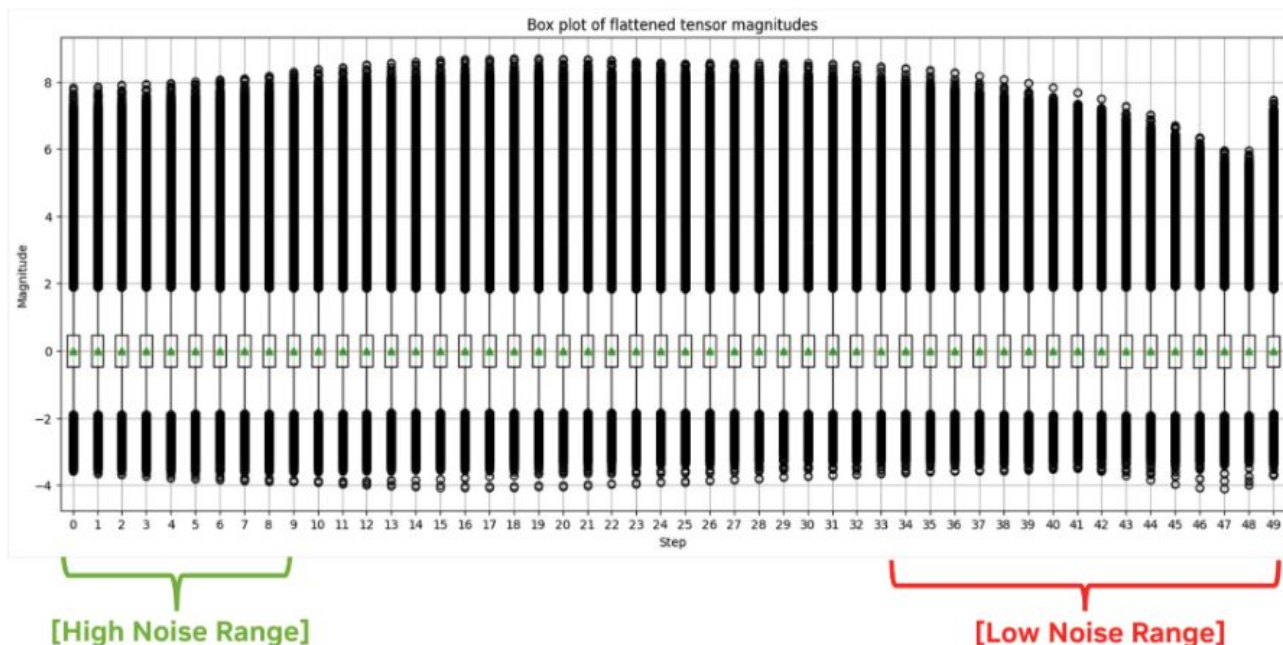
- Details [Low noise range]





# Why TensorRT's quantization approach is unique

TensorRT's *Percentile Quant* focuses on the important percentile of the noise range by determining the minimum quantization scaling factors from the selected noise range. We observe that outliers are not very sensitive to the image quality.







**Visual and performance benchmark results**

# Visual benchmark results

TensorRT 8-bit quantization generates near-identical images compared to FP16 baseline



FP16

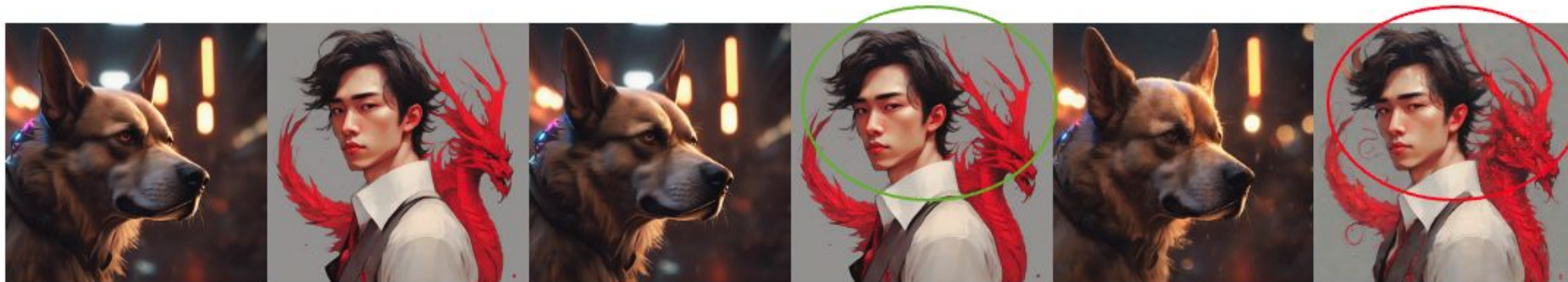
TensorRT INT8

TensorRT FP8



## A closer look at the images

TensorRT 8-bit quantization generates near-identical images compared to FP16 baseline



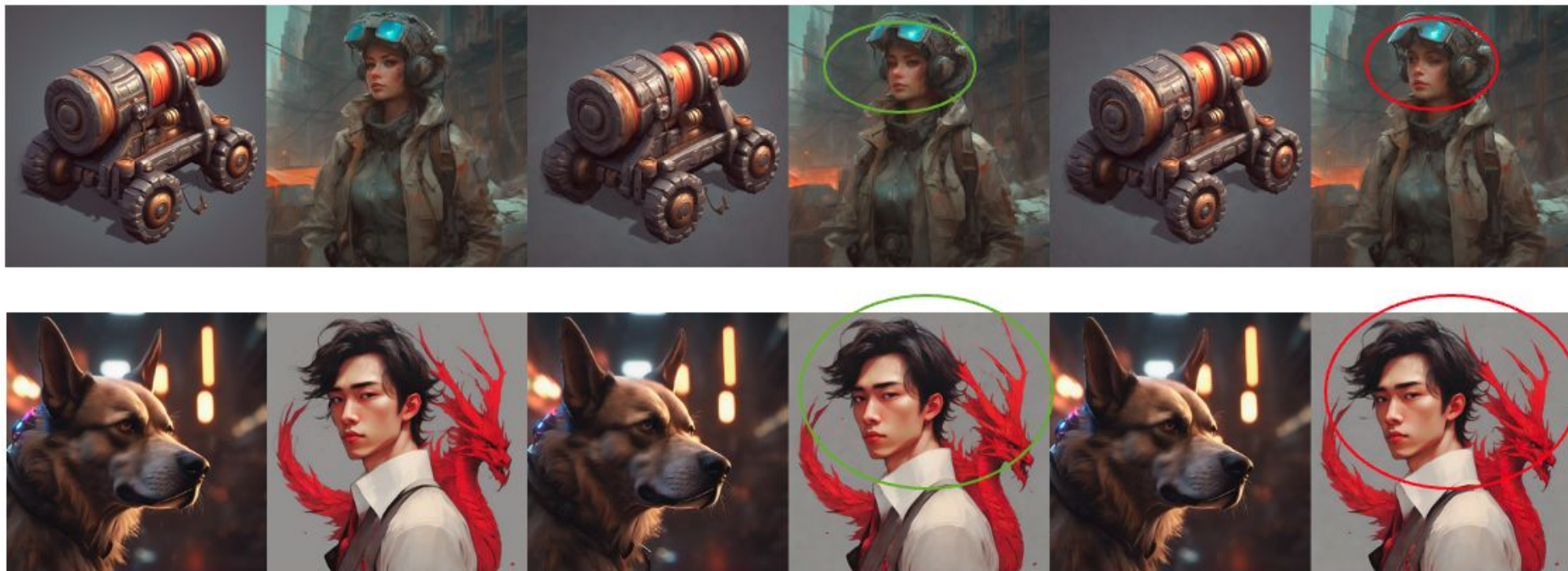
FP16

INT8 with Nvidia TensorRT Percentile Quant

INT8 with original SmoothQuant

## A closer look at the images

TensorRT 8-bit quantization generates near-identical images compared to FP16 baseline



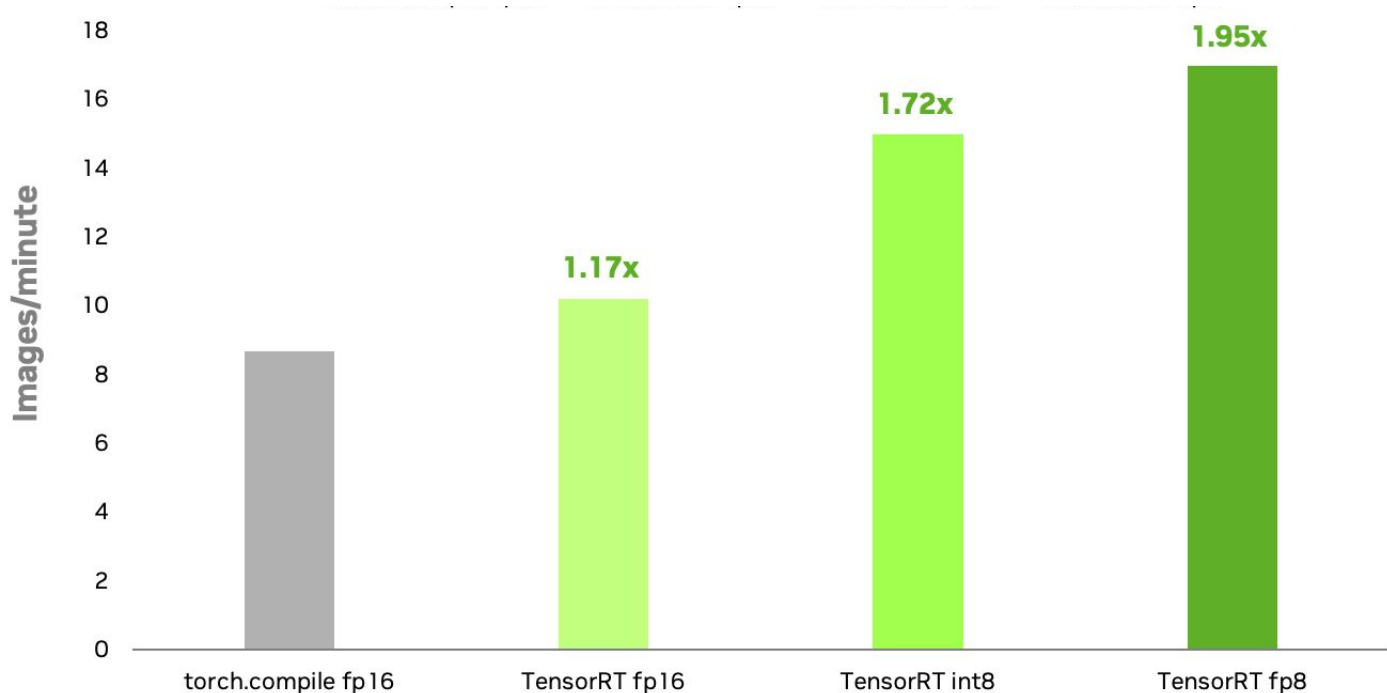
FP16

FP8 with Nvidia TensorRT Percentile Quant

FP8

# Performance Benchmark

Performance measurement on RTX 6000 Ada



Configuration: Stable Diffusion XL 1.0 base model; images resolution=1024×1024; Batch size=1; Euler scheduler for 50 steps. TensorRT INT8 quantization is available now, with FP8 expected soon. The benchmark for TensorRT FP8 may change upon release. The additional speedup of FP8 over INT8 is primarily attributed to the quantization of MHA layers.





**E2E workflow demo: Speedup SDXL inference  
with TensorRT's quantization toolkit**

## E2E workflow Demo and tech blog

- Live demo: Run Jupyter Notebook in VS Code. Download [here](#).
- Check out our blogpost: [NVIDIA TensorRT Accelerates Stable Diffusion Nearly 2x Faster with 8-bit Post-Training Quantization](#)
  - [Twitter](#) | [LinkedIn](#) | [Facebook](#) | [Instagram](#) | [NVShare](#)



NVIDIA AI

738,847 followers

2d • 🌐

+ Follow ...

Technical deep dive: [#NVIDIATensorRT](#) optimization significantly enhances stable diffusion [#inference](#) speeds by a factor of 2, resulting in improved performance for low-latency applications ➡ <https://nvda.ws/3lyfnLR>



FP16

TensorRT INT8

TensorRT FP8

# TensorRT diffusers preview

Enabled by AMMO

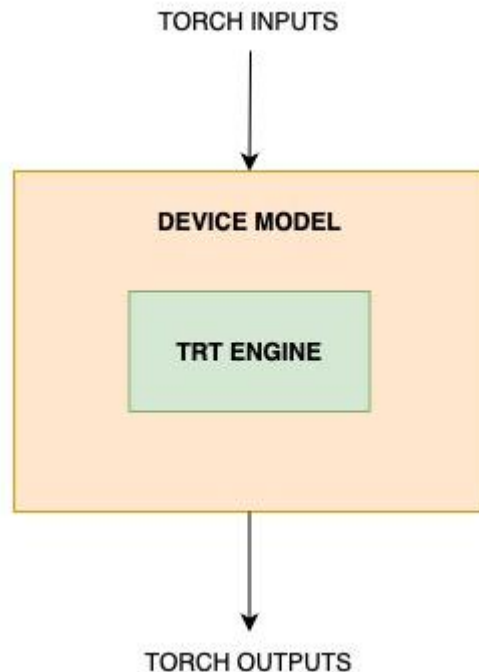
- Models support
  - SDXL
  - SDXL-turbo
  - SD 1.5
  - SVD (coming soon post GTC)
  - DiT (coming soon post GTC)
- Low-bit precision support
  - INT8
  - FP8
- Diffusion Caching (coming soon post GTC)
- **Diffusion Runner** (coming soon post GTC)



## Preview Feature: Diffusion Runner

Easy to use tool for generating images with diffusion models

- TensorRT is a complex system including graph optimization, compiler, online kernel benchmarking, and more.
- First-time TensorRT users need time to learn the the TensorRT workflow
- DiffusionRunner is created to reduce the TensorRT learning curve by encapsulating the TensorRT engine and runtime as the **DeviceModels**
  - **DeviceModels** offers torch-like inference API and experience with TensorRT speed.
- DiffusionRunner supports TensorRT engine building & loading of the quantized diffusion models and diffusion inferences



## Diffusion Runner example

```
from ammo.torch.deploy.diffusion.diffusion_runner import DiffusionRunner

# Create the runner with the quantized checkpoint
runner = DiffusionRunner(unet=<quantized torch model>)

# Build the runner once, optional to supply a quantized onnx or trt model
runner.build()

# Run inference multiple times for the desired text prompts
runner.infer(["a beautiful photograph of Mt. Fuji during cherry blossom"])
runner.infer(["A white goose holding a paint brush"])
runner.infer(["A modernist courtroom in a rainforest by Raphael"])
```

## Summary

- Quantization is a very effective model optimization technique for LLMs and diffusion models
- Quantization reduces model size, speeds up inference, and lowers cost
- NVIDIA's model optimization platform (AMMO) provides easy & accurate APIs to quantize generative AI models and deploy to TensorRT/TensorRT-LLM

The background is a dark, almost black, field filled with a complex network of thin, glowing green lines. These lines connect various points, some of which are highlighted as bright green dots. The overall effect is reminiscent of a neural network diagram or a data visualization of connections. The word "Questions?" is written in a large, white, sans-serif font, positioned in the upper left quadrant of the image.

# Questions?

Asma Beevi KT, [akuriparambi@nvidia.com](mailto:akuriparambi@nvidia.com)  
Zhiyu Cheng, [zhiyuc@nvidia.com](mailto:zhiyuc@nvidia.com)

