Enable Blackwell Inference With TensorRT Model Optimizer

Huizi Mao, Jingyu Xin, Ye Yu

03/17/2025

Outline

1. Overview of TensorRT Model Optimizer (ModelOpt)
2. ModelOpt quantization
   a. FP4 for large language models
   b. FP4 for diffusion models & others
3. ModelOpt speculative decoding

ModelOpt - A Toolkit for Inference Optimization

- Inference optimization algorithms
- APIs to apply optimizations for HF/NeMo/Megatron models
- Gateway to deployment optimized solutions - TensorRT-LLM, TensorRT, vLLM etc

Install ModelOpt library:

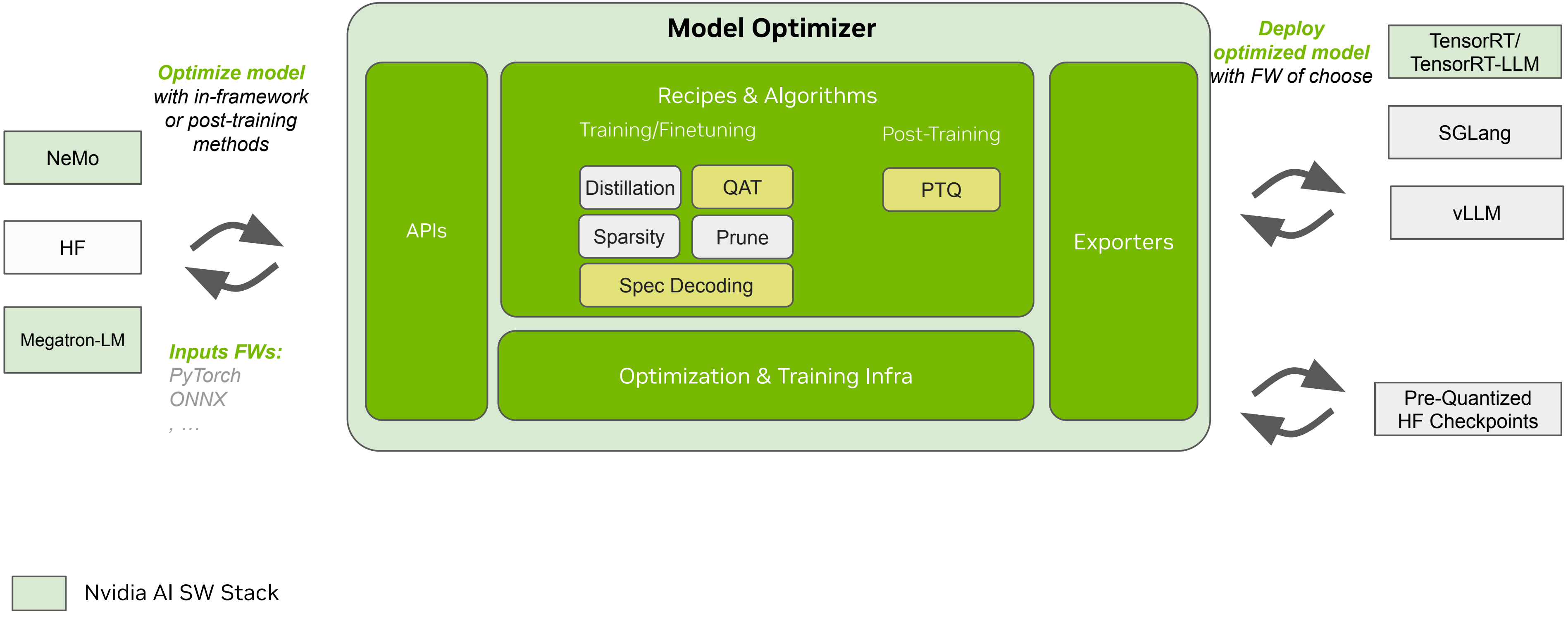
```
pip install nvidia-modelopt --extra-index-url https://pypi.nvidia.com
```

OSS ModelOpt library and examples on GitHub:

https://github.com/NVIDIA/TensorRT-Model-Optimizer

NVIDIA TensorRT Model Optimizer (ModelOpt)

Product Overview



Enable FP4 inference for LLMs

Quantization Formats Supported by ModelOpt

Quantization format	Details	GPU support
INT8	• Weight & act quantization • ~2x memory and compute saving	All
FP8	• Weight & act quantization • ~2x memory and compute saving, near lossless	Ada, Hopper & Blackwell
INT4 (weight-only)	• INT4 weights, FP16 activations • ~4x memory saving, no compute saving	All
FP4	• Weights & act quantization • ~4x memory and compute saving	**New on Blackwell**
Mixed FP4	• FP4 weights, FP8/BF16/etc activations • ~4x memory saving, compute saving depends	**New on Blackwell**

Comparing FP4 with other formats

- Advantages of Blackwell FP4 formats
  - **8-bit block scaling factors** -> less overhead, finer-grained block sizes than INT4
  - Allows **two-level scaling** -> larger dynamic range
  - Native hardware support for matmul and decoding

ModelOpt Post-training quantization (PTQ) Workflow

- PTQ is a simple way to get started for FP4 inference

Pytorch model

Calibrate and quantize

```
mtq.quantize()
```

Export TRT-LLM checkpoint /
HF checkpoint / ONNX graph

```
mte.export_hf_checkpoint()
mte.export_tensorrt_llm_checkpoint()
```

Build and serve models via
TRT-LLM / vLLM / SGLang / TRT

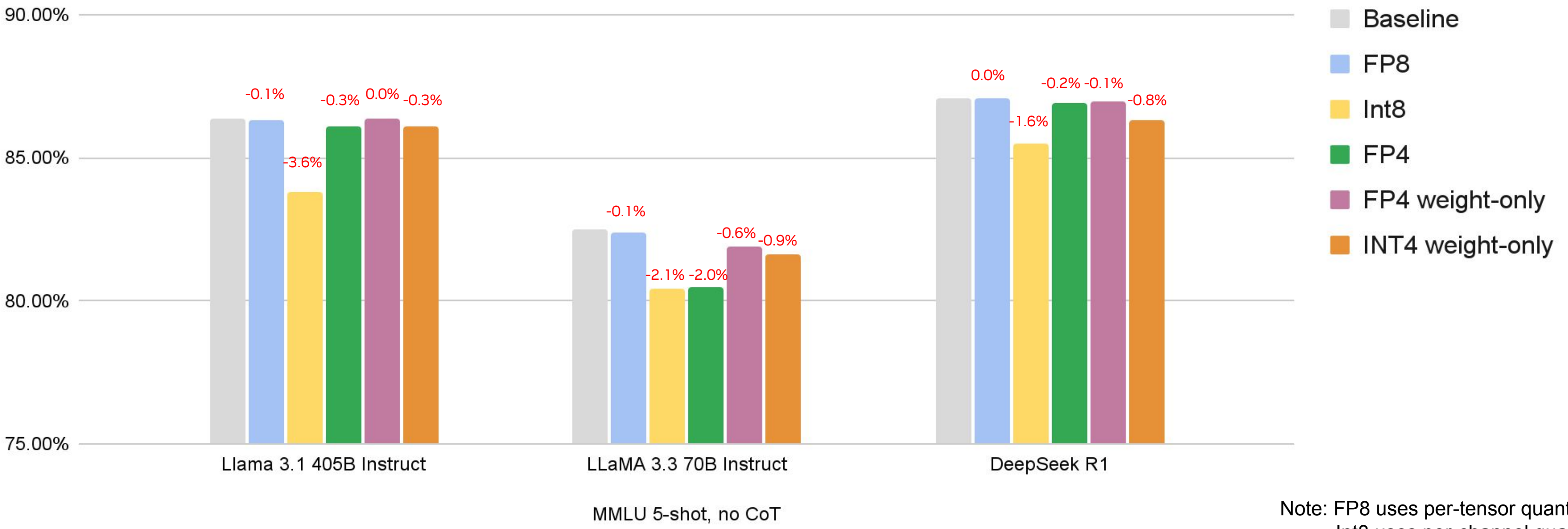
```
tensorrt_llm.llmapi.llm.LLM()
python -m vllm.entrypoints.api_server
python -m sglang.launch.local_server
```

Quantization Flow in ModelOpt - FP4 PTQ

```
import modelopt.torch.quantization as mtq
import modelopt.torch.export as mte

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
model = mtq.quantize(model, mtq.NVFP4_DEFAULT_CFG, forward_loop)


# Deploy the model via TRT-LLM/etc
mte.export_hf_checkpoint(model, export_dir=export_path)
```

Comparing PTQ accuracy

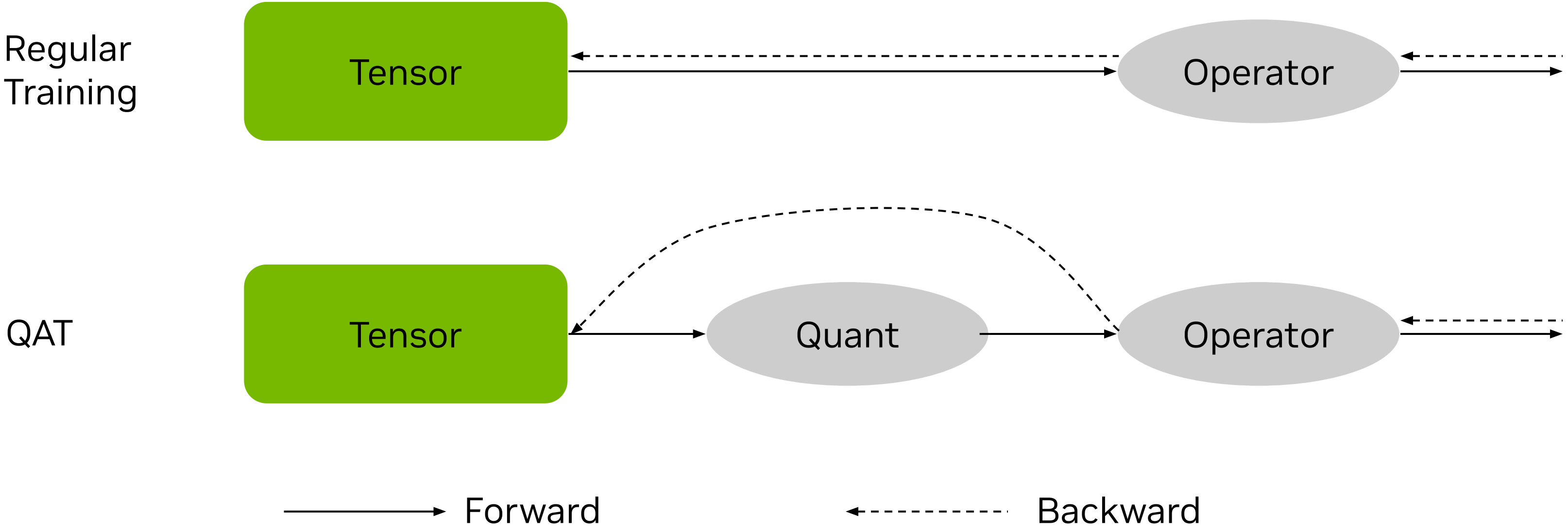
- FP4 shows decent accuracy in **weight & activation** quantization
  - Better accuracy than INT4 in **weight-only** quantization



Note: FP8 uses per-tensor quantization
Int8 uses per-channel quantization

Quantization-aware Training (QAT)

- QAT can further improve the quantization accuracy
  - A simple yet effective QAT method is straight-through estimator (STE)
  - QAT (different from Quantized Training) requires weights in original precision



Quantization Flow in ModelOpt - FP4 QAT

- QAT can be simply done on the PTQ'd model

```
<cont'd from Slide 9>

# Perform PTQ: Add quantizer nodes and perform calibration
model = mtq.quantize(model, atq.NVFP4_DEFAULT_CFG, forward_loop)

# Do finetuning(QAT) with the original training loop
# Epoch number/Learning rate should be reduced
train(model, dataloader, loss_func)

# Save the weights and quantization state for evaluation or resumed training
mto.save(model, filename)
```

Better accuracy with QAT

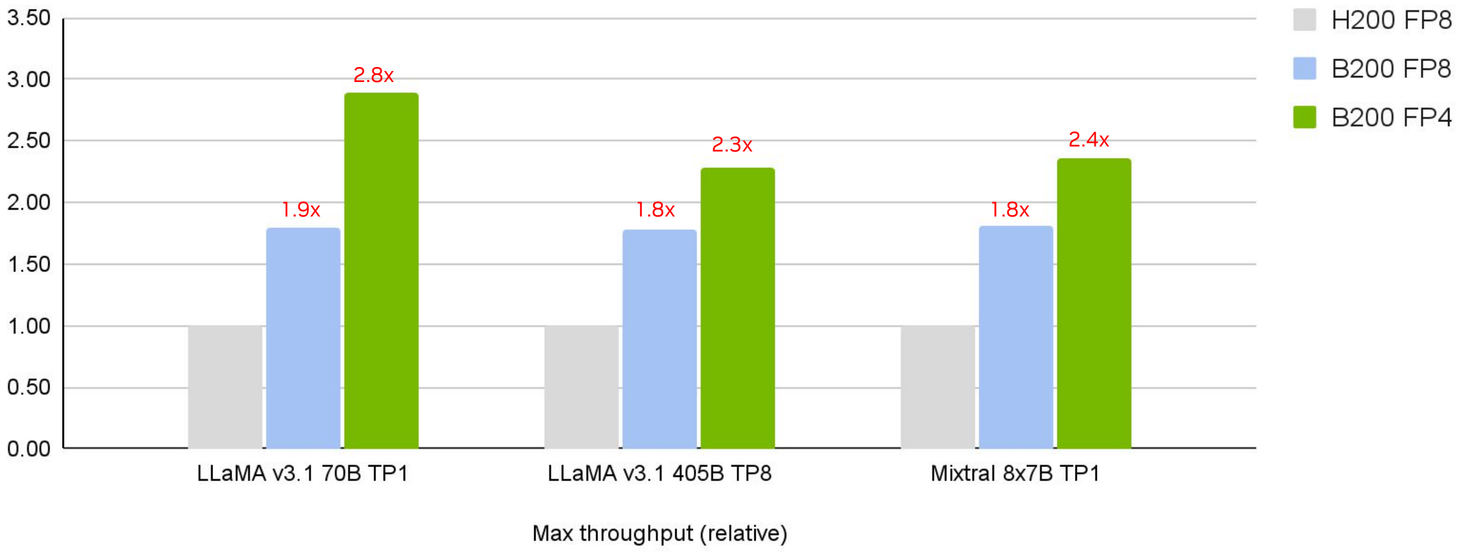
- FP4 QAT on original training data can fully recover accuracy
  - Results obtained with NVIDIA's Nemotron 4 with internal training data and pipeline
  - ~5% pretraining data is used

Method	Nemotron-4-15B base	Nemotron-4-340B base
BF16 (baseline)	64.2%	81.1%
FP4 with PTQ	61.0%	80.8%
FP4 with QAT	**64.5%**	**81.4%**

5-shot MMLU accuracy

FP4 Performance on Blackwell

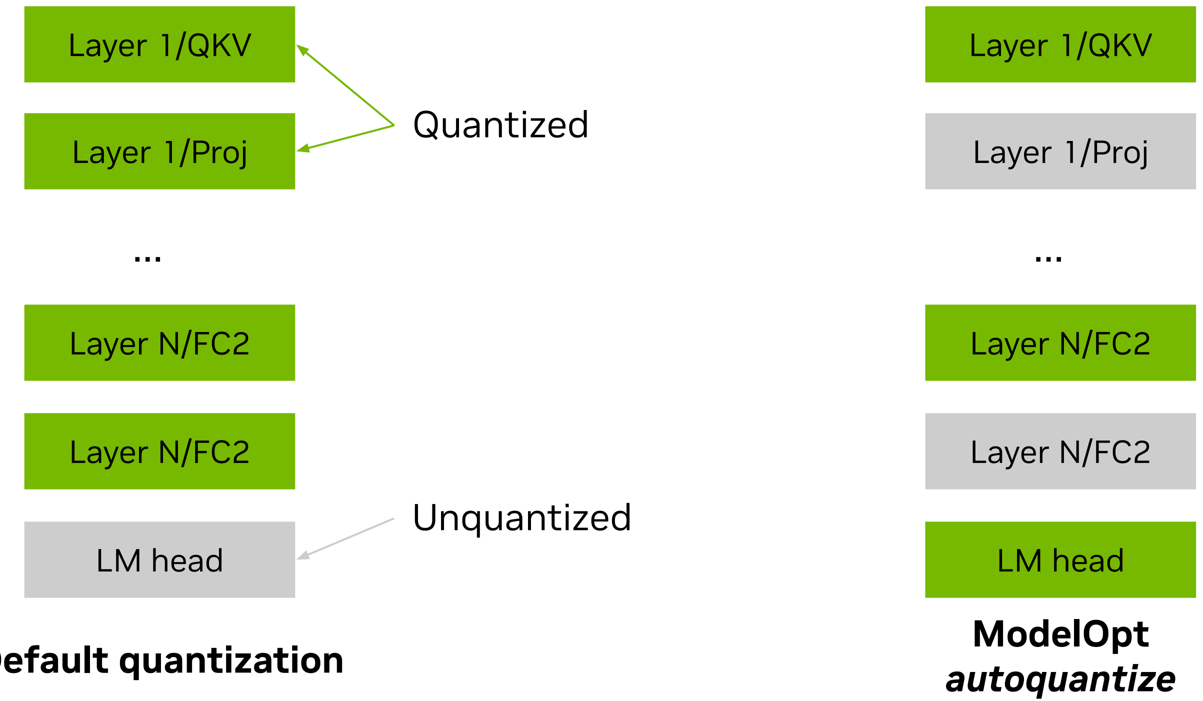
- FP4 greatly improves LLM inference performance
  - Initial data below, more optimizations coming soon



Results measured with TRT-LLM v0.17

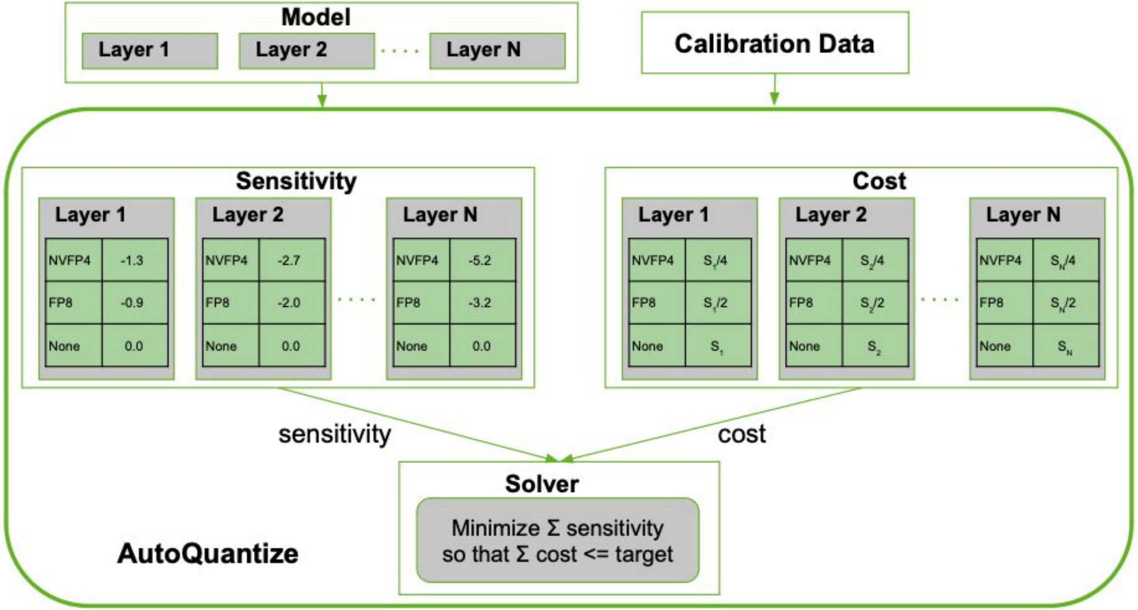
Mixed layer precision quantization

- Mixed layer precision - keep sensitive layers in high precision
  - Supported via ModelOpt's *autoquantize* and TRT-LLM

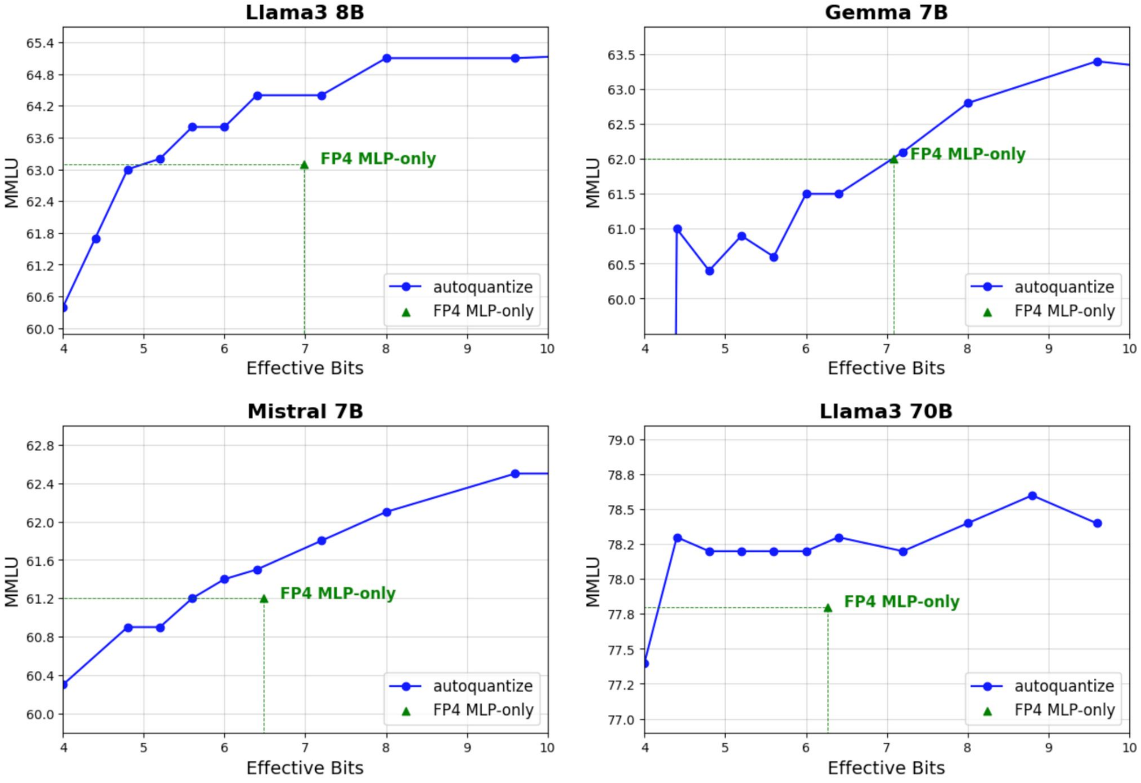


AutoQuantize - Automatic mixed layer precision

- Given compression target, find layers to minimize sensitivity (accuracy impact proxy)



Results: Quantized Model with AutoQuantize



- AutoQuantize is usually better than human heuristics

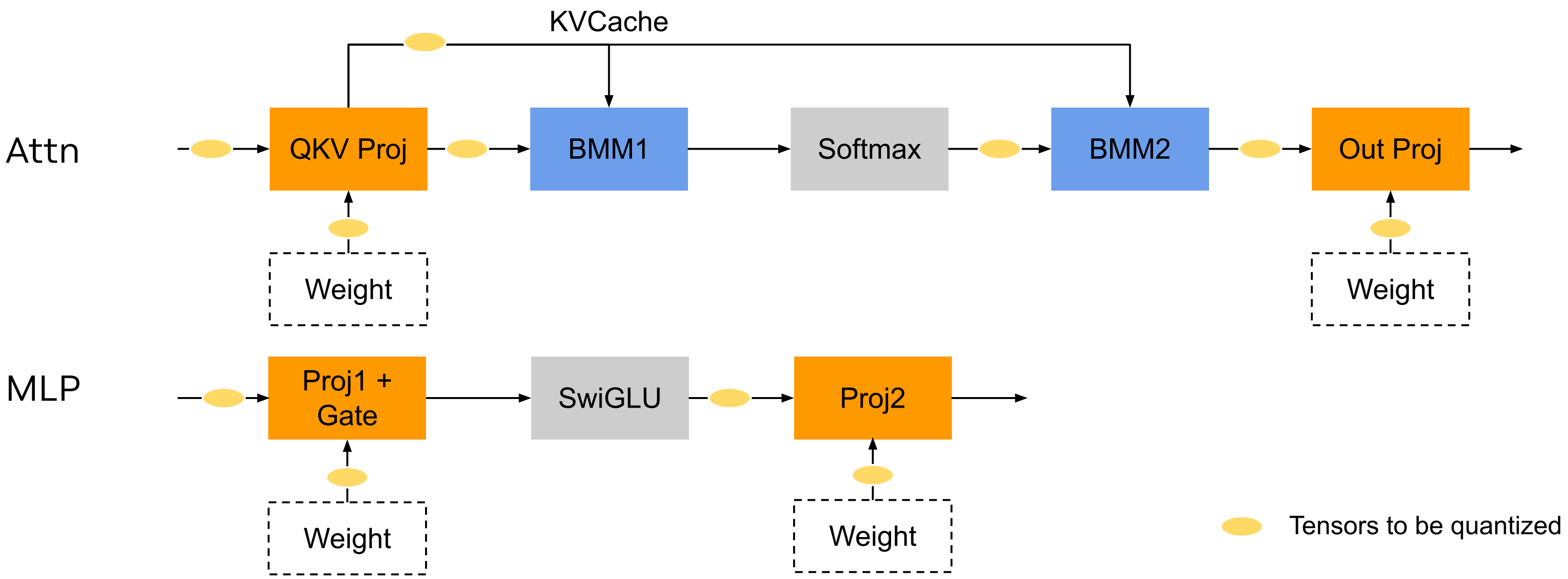
- How to use AutoQuantize in ModelOpt:

  `mtq.auto_quantize()`

FP4 MLP-only - A common recipe that quantize MLP linear layers to FP4, leave others to FP8

Advanced topic - Mixed tensor precision

- **Weight quantization** -> memory size/bandwidth (all scenarios)
- **Act quantization** -> GEMM math throughput (prefill & large-batch decode)
- **KV Cache quantization** -> memory size/bandwidth (long-context/large-batch decode)
- **BMM quantization** -> Attention math throughput (long-context prefill)



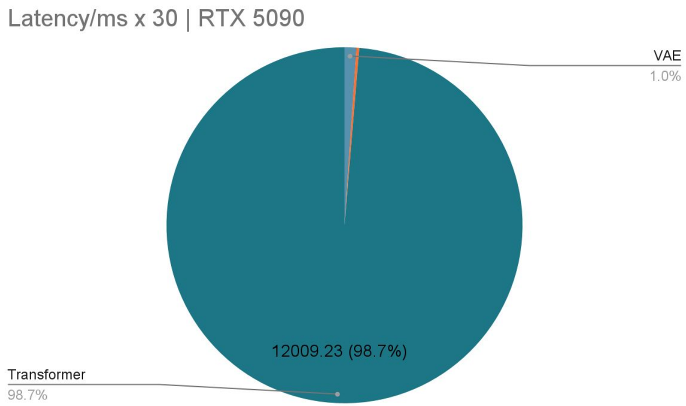
Mixed tensor precision features in ModelOpt/TRTLLM

- Mixed precision GEMM kernels
  - Int4 weight, FP8 activation, math in FP8 -> `mtq.W4A8_AWQ_BETA_CFG`
  - FP4 weight, FP8 activation, math in FP8 (coming soon)
- Mixed precision Attention kernels
  - FP8 KV Cache -> `mtq.FP8_WA_FP8_KV_CFG`
  - FP8 Attention math -> `--use_fp8_context_fmha` (TRTLLM flag)
  - FP4 KV Cache, FP8 Attention math (coming soon)

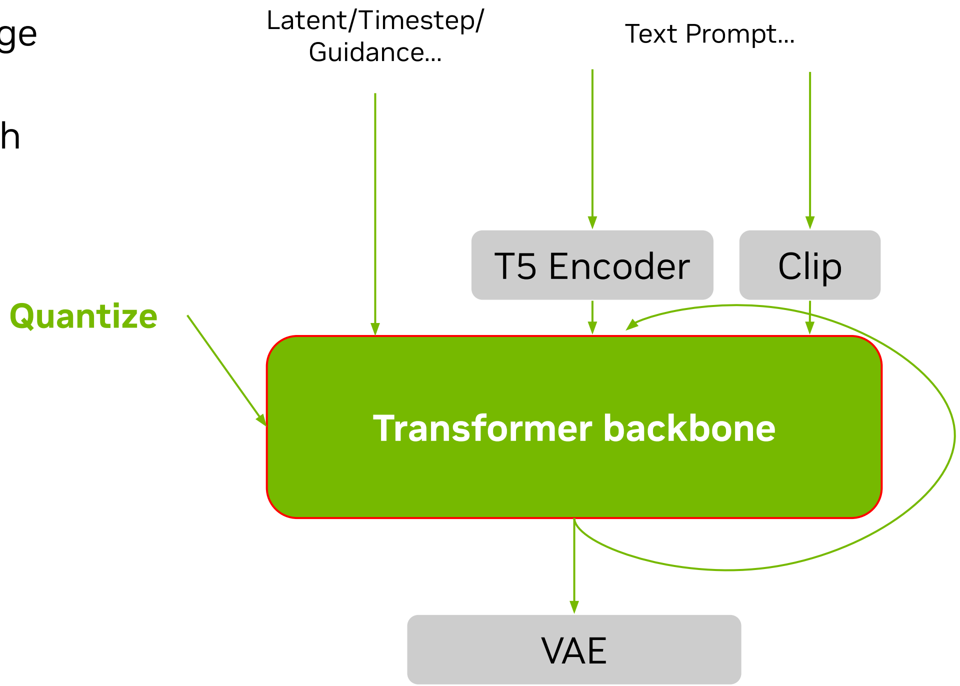
Enable FP4 inference for Diffusion models

Flux.1 Dev FP4 Workflow

- **Flux.1 Dev**, one of the state-of-the-art image generation models
- Goal: optimize the backbone with FP4, which drives **98%+** of overall latency



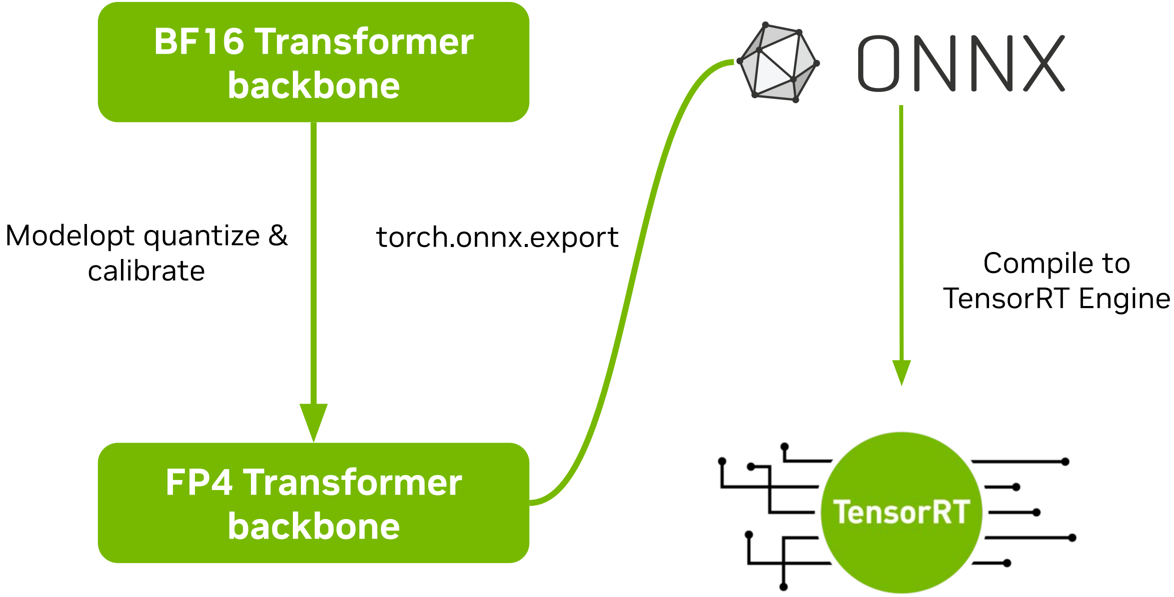
Latency breakdown



Flux.1 Dev architecture

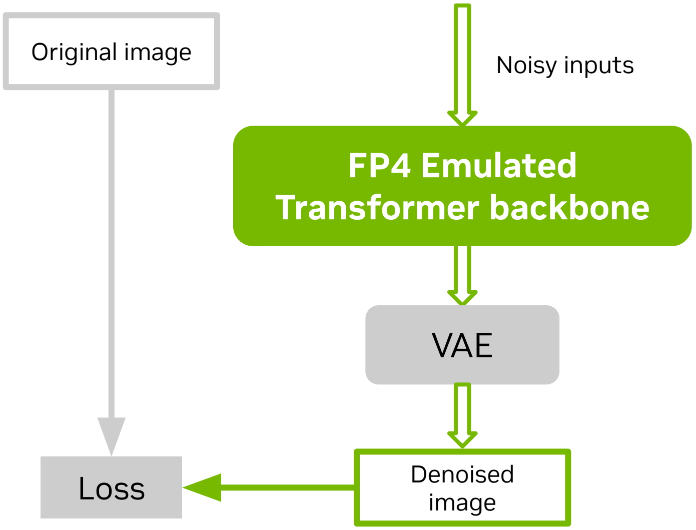
Flux.1 Dev FP4 Workflow

- ModelOpt Supports calibration of any PyTorch-based model
- Deploy using ONNX-TRT or other compatible solutions

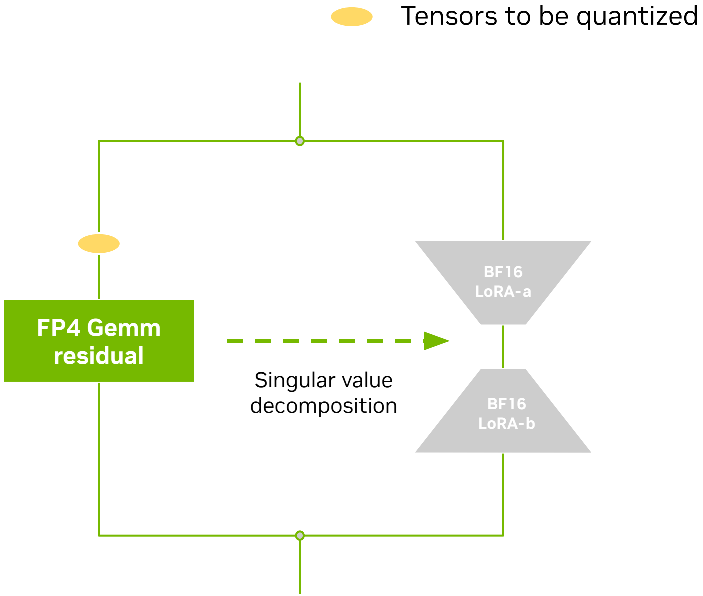


Quantization-aware Training (QAT) and SVDQuant

- FP4 PTQ can cause artifacts (e.g., awkward text) sometimes and lower metrics vs. BF16.
- QAT, or with SVDQuant can lead to higher accuracy.



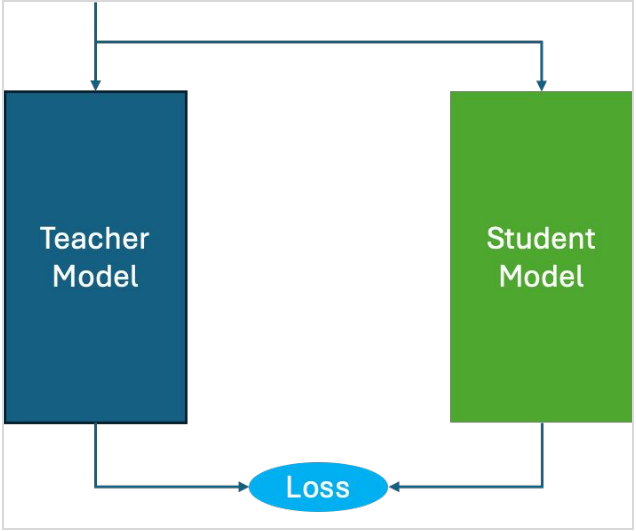
QAT training on Diffusion



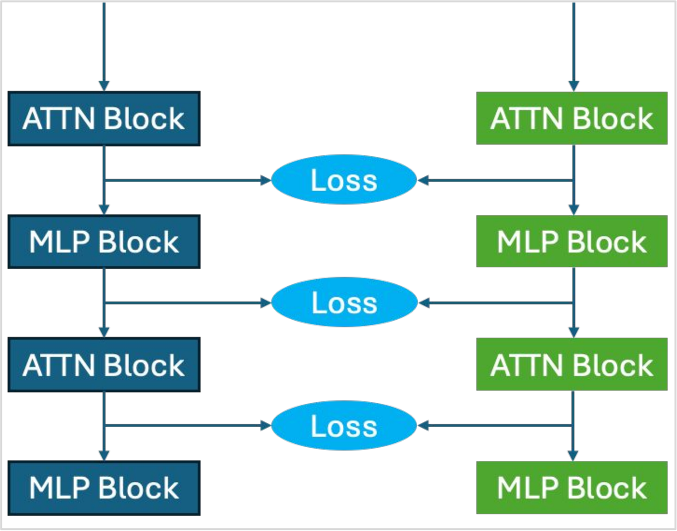
SVDQuant explanation

Distillation with QAT

- Standard QAT
  - Due to the synthetic data, regular QAT is challenging
- Distillation-Based QAT
  - Used BF16 teacher and quantized student.
  - Boosted image quality via output and block-wise distillation



Output Distillation



Block-wise Distillation

FP4 Results: Quality

- Demonstrated notable improvements in quality and clarity, based on example images and metrics
- QAT expects further gains by using original Flux training data instead of synthetic data

Numerical results for FP4

Model	Image Reward	CLIP-IQA	CLIP
BF16	1.118	0.926	30.150
FP4 PTQ	1.096	0.923	29.860
FP4 QAT	**1.119**	**0.928**	**29.920**
FP4 SVDQ	**1.108**	**0.927**	**30.068**

BF16

FP4 PTQ

Prompt: A pack of dogs gather around a laptop, intently watching a tutorial on digital painting. The screen displays the words "Blend colors effectively."

FP4 QAT

FP4 SVDQuant

FP4 Results: Performance

- Use FP4 inference on Blackwell GPUs to reduce VRAM and latency.
- Switch to Low-VRAM Mode in DemoDiffusion if VRAM is limited (higher latency).

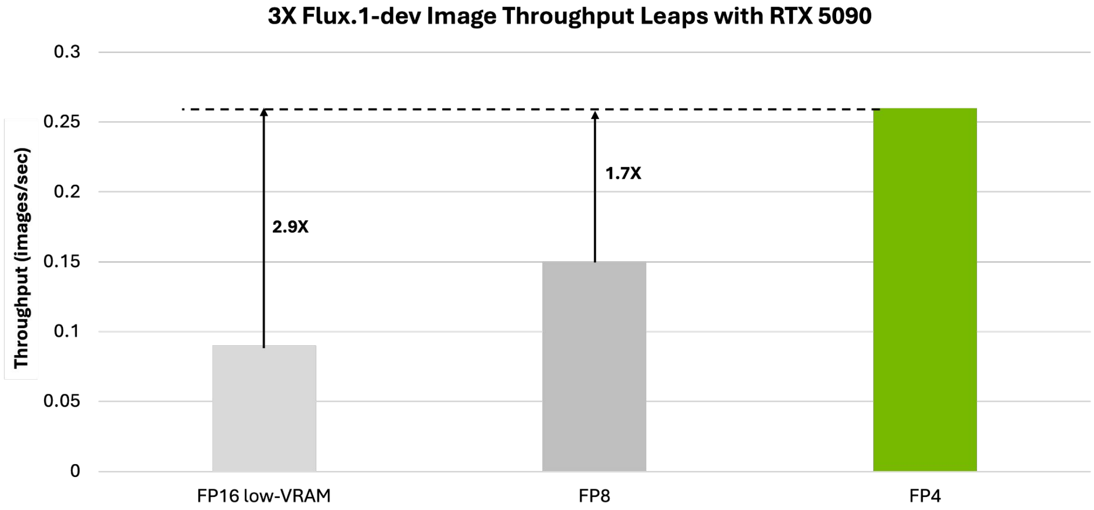


Diagram showing 3X Flux. 1-dev image throughput leaps with RTX 5090 with FP4 achieving 2.9X greater throughput than FP16 low-VRAM and 1.7x greater throughput than FP8

Appendix: FP4 on Video-Gen Models

Cosmos-7B, NVIDIA's STOA world foundation model, sees video quality improvement with FP4 SVDQuant.

BF16

FP4 PTQ

FP4 SVDQuant

Screenshot of the generated video

Appendix: Live demo

Code demo && demoDiffusion Inference on RTX 5090

```
+ # Configuration for knowledge distillation (KD)
+ kd_config = {
+     "teacher_model": teacher_model,
+     "criterion": distill_config["criterion"],
+     "loss_balancer": distill_config["loss_balancer"],
+     "expose_minimal_state_dict": False,
+ }
+ transformer = mtd.convert(transformer, mode=[("kd_loss", kd_config)])

# Move the model to the appropriate device and set the desired weight precision
transformer.to(accelerator.device, dtype=weight_dtype)
transformer.requires_grad_(True)

# Making sure to freeze the weights from model._teacher_model
transformer, optimizer, train_dataloader, lr_scheduler = accelerator.prepare(
    transformer, optimizer, train_dataloader, lr_scheduler
)

# Compute the distillation loss using ModelOPT's compute_kd_loss
+ ...
+ loss = transformer.compute_kd_loss(...)
+ ...
```

Calibration && QAT code is available at [ModelOPT Github Repo].

Broader Model/Platform support via ONNX

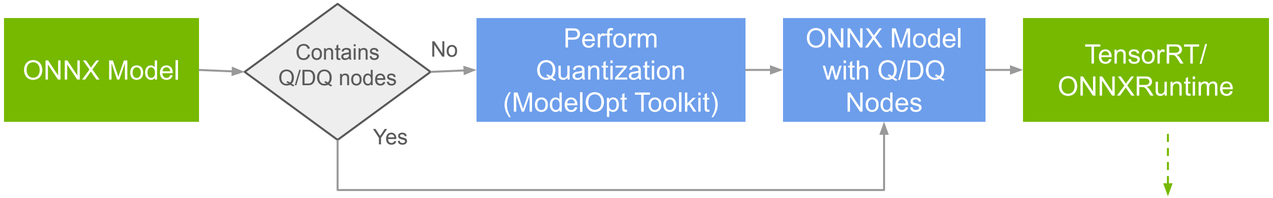
modelopt.onnx subpackage - General support & Platform agnostic

- Natively supports diverse model architecture beyond language transformer
  - ResNets
  - ViT, SwinT
  - BEVFormer, BEVFusion
  - Llama family
- Supports a broad range of platforms
  - Windows, Desktop Linux, Mobile Linux

ONNX Quantization Workflow

- ModelOpt ONNX quantization recognizes graph pattern and explicitly inserts Q/DQ nodes
    - Better user control over TensorRT’s implicit quantization
- Currently supports FP8/INT8/INT4

```
$ python -m modelopt.onnx.quantization --onnx_path=model.onnx --quantize_mode fp8
```




```
$ trtexec --onnx=model.quant.onnx --fp8
```

Low-latency inference with ModelOpt

Speculative decoding

Speculative decoding

- More applications demand ultra low latency in LLM generation
- LLM inference is usually memory bound
- Speculative decoding computes multiple tokens in one generation, identical output distribution

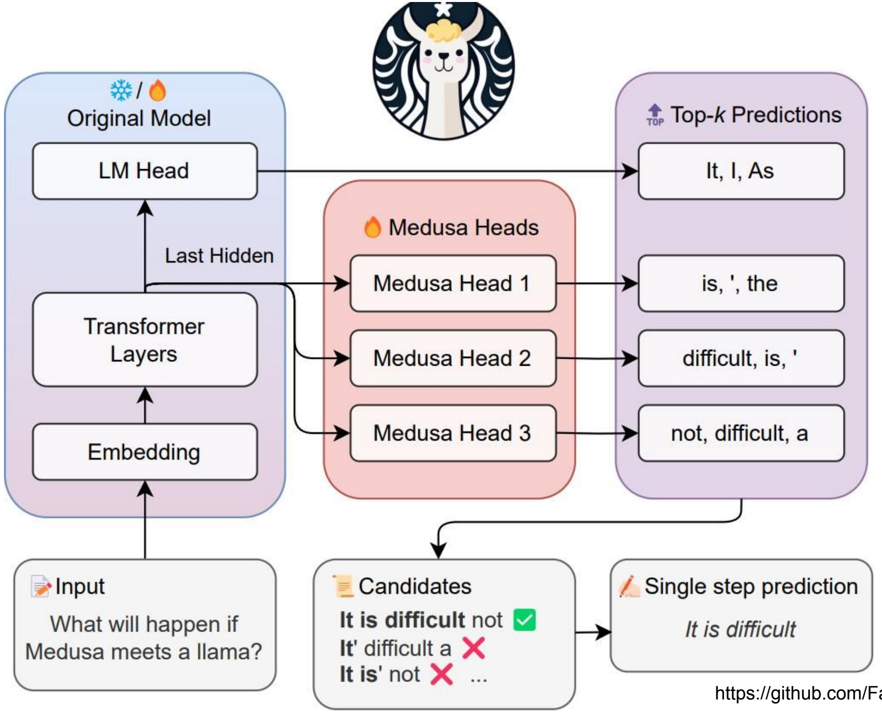
https://research.google/blog/looking-back-at-speculative-decoding/

Supported speculative decoding algorithms

Algorithm	Details	Framework	Deployment
Medusa	• Finetuned Medusa heads • Predict multiple future tokens in parallel	• Huggingface • Megatron-LM • NeMo (coming soon)	• TensorRT-LLM • vLLM (coming soon)
EAGLE	• Finetuned EAGLE module • Predict multiple future tokens autoregressively	• Huggingface • Megatron-LM • NeMo (coming soon)	• TensorRT-LLM • vLLM (coming soon)
Multi-Token Prediction	• Finetuned MTP modules • Predict multiple future tokens autoregressively	• Megatron-LM • Huggingface (coming soon) • NeMo (coming soon)	• TensorRT-LLM

Medusa

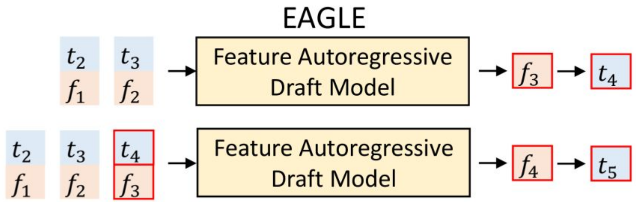
- Multiple decoding heads for parallel future tokens predictions

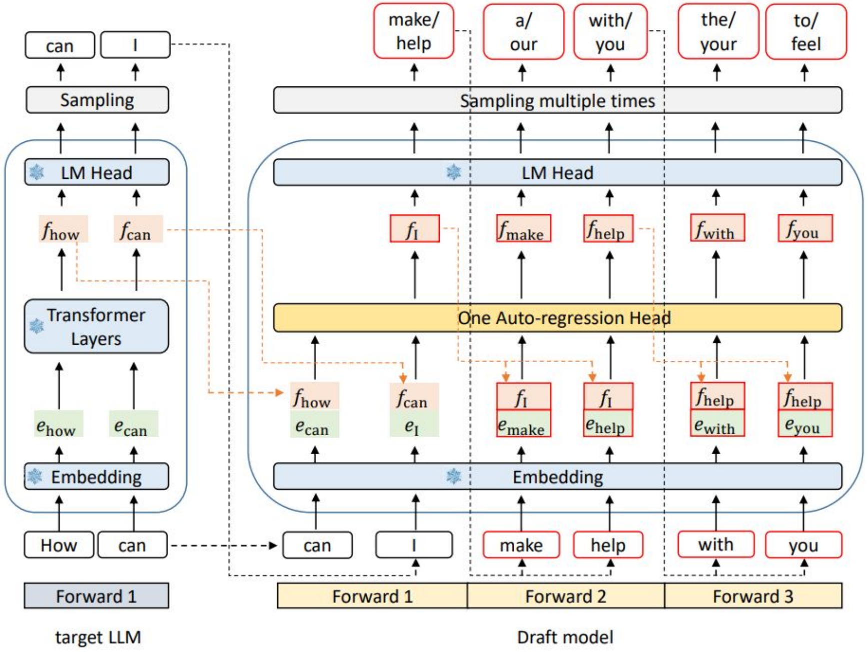


https://github.com/FasterDecoding/Medusa

EAGLE

- An autoregressive draft module to predict future tokens
- Predict in feature space

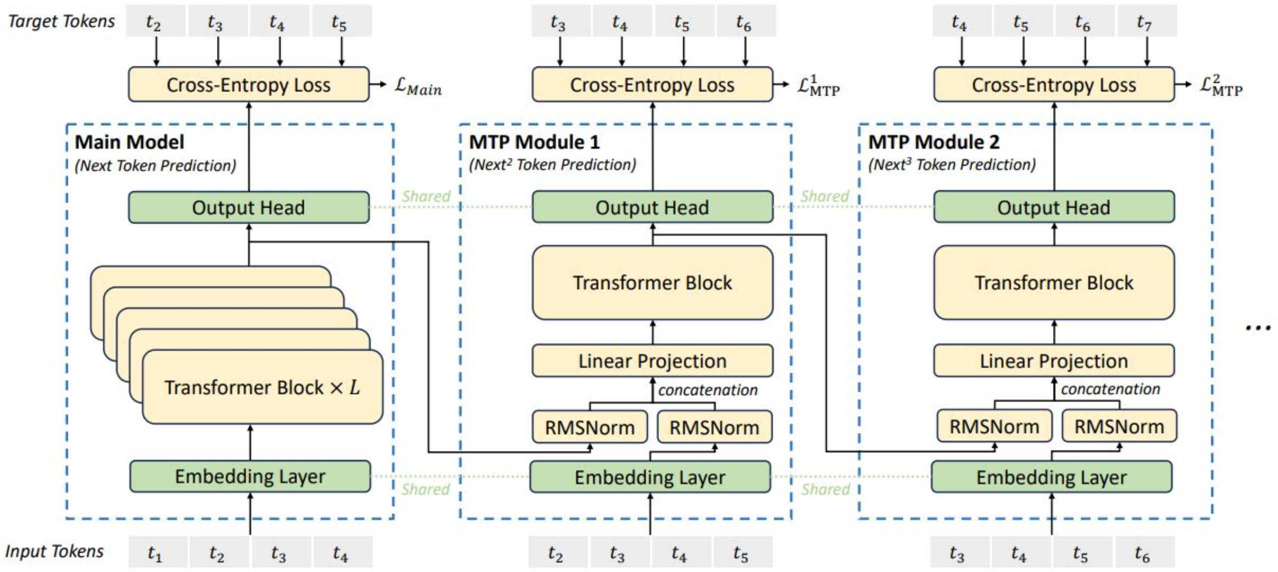




https://arxiv.org/html/2406.16858v1

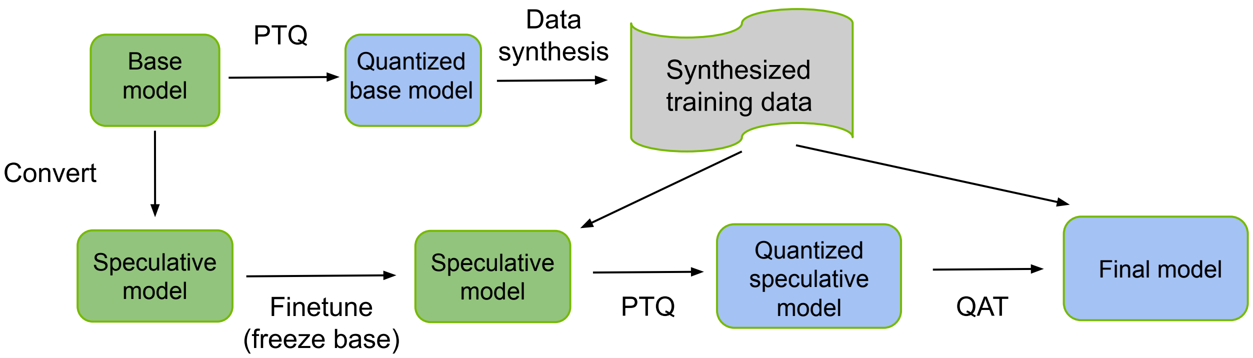
Multi-Token Prediction

- Similar to EAGLE in architecture
- Use different modules for multiple token predictions



https://arxiv.org/pdf/2412.19437

Speculative model training workflow

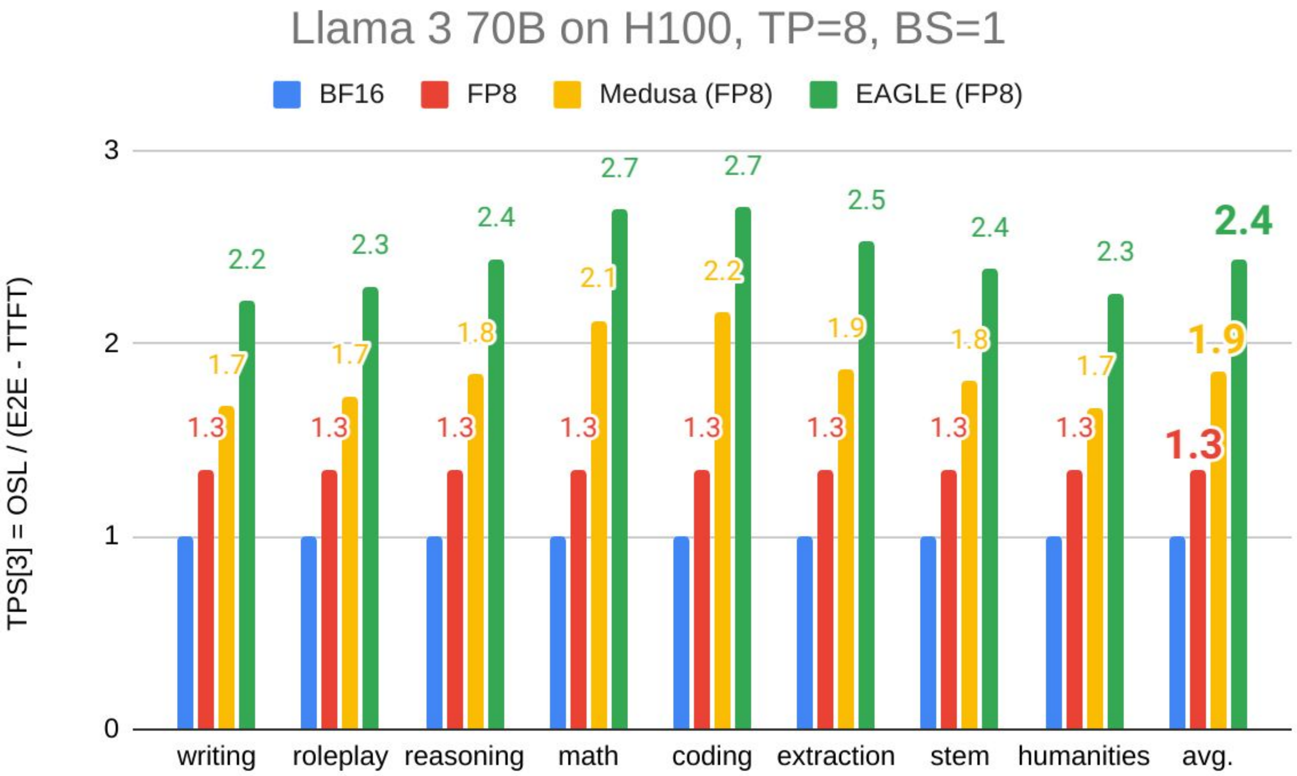


- (Optional) Quantize base model
- Synthesize training data using (quantized) base model
- Convert base model into speculative model
- Finetune speculative model (optional: freeze base model)
- (Optional) Quantize speculative model (PTQ)
- (Optional) Finetune quantized speculative model (QAT)
- (Optional) Pruning and sparsifying

Demo

```python
import transformers
import modelopt.torch.speculative as mtsp


model = transformers.AutoModelForCausalLM.from_pretrained("meta-llama/Llama-3.2-1B-Instruct")
config = {
  "eagle_num_layers": 1,
}
mtsp.convert(model, [("eagle", config)])
```

Perf results

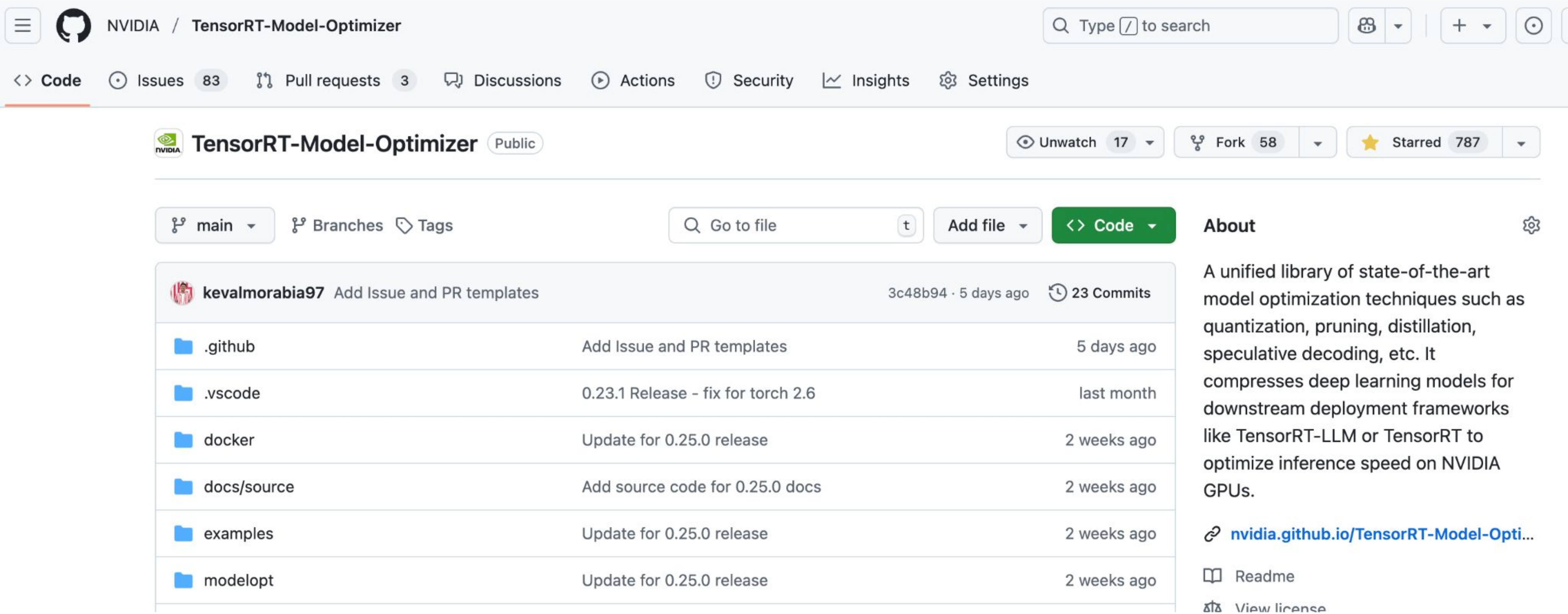


Summary

- Blackwell boosts GenAI inference speed with better hardware and advanced features
- Model Optimizer toolkit provides a gateway for inference optimization on NV ecosystem
    - FP4 quantization
    - Speculative decoding
    - And more features from pytorch to deployment

Acknowledgement: Omri Almog, Chenjie Luo, Zhiyu Cheng, Kai Xu, Asma KT Kuriparambil, Simon Layton, etc

ModelOpt is now open-source on Github



https://github.com/NVIDIA/TensorRT-Model-Optimizer

Questions?

NVIDIA