美团PyTorch量化工具

设计哲学、核心特性及性能基准

李庆源 李亮 张勃

美团/基础研发平台

大纲

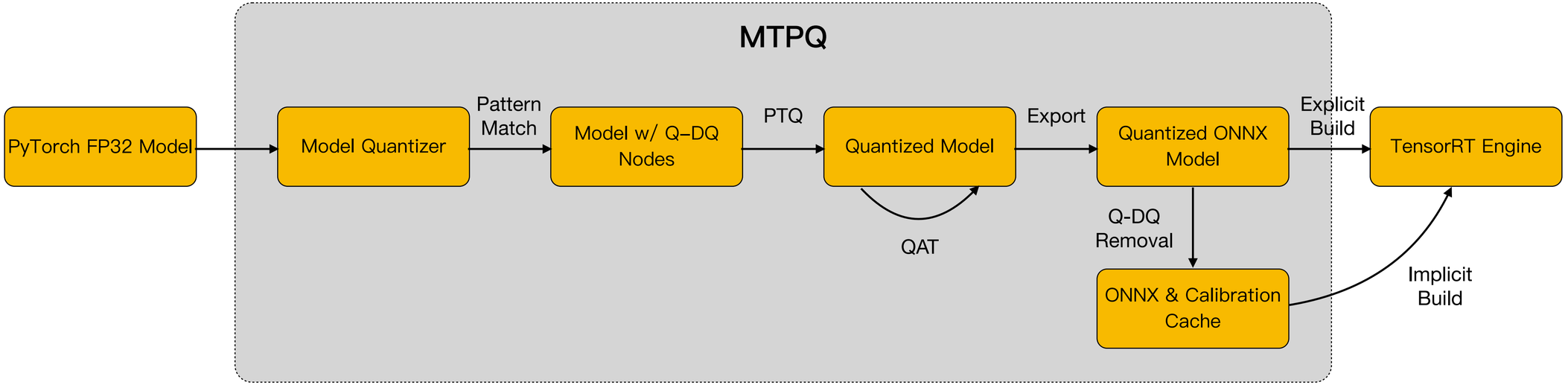
- 为什么要设计 MTPQ?
- 概述
  - 量化流程
  - 软件架构
  - 使用示例
- 核心特性
  - 量化算子自动化插入
  - 高精度量化算法集成
  - 可扩展性和框架支持
  - 大语言模型的量化
- 性能基准测试
- 展望

为什么要设计 MTPQ?

- **现有问题:** 实际应用场景的量化难点
  - 后量化（PTQ）：精度难以保证
  - 训练感知量化（QAT）：部署不友好，没有为特定硬件和部署框架设计
- **解决方法：** 对 PyTorch 模型量化过程进行专门的优化，适配主流推理框架（如 TensorRT ）的量化部署

概述 – 量化流程

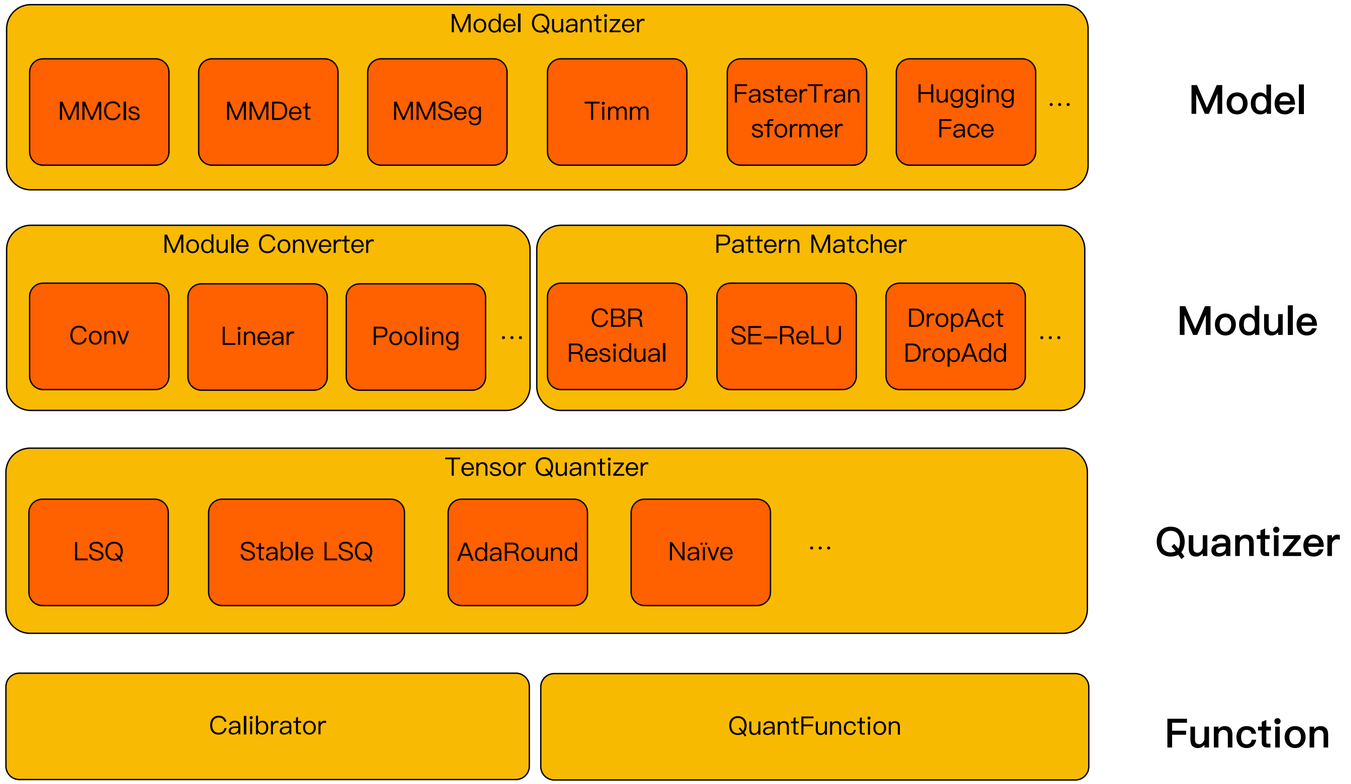
高效且部署友好的量化流程



概述 – 软件架构

自顶向下模块化设计，即插即用，支持自定义

内置 OpenMMLab、Timm、FasterTransformer、HuggingFace 等主流仓库的量化器



概述 – 示例使用

使用 MTPQ，只用几行代码就可实现量化功能

```
from timm.models import create_model

from mtpq.model_quantizer import ModelQuantizerFactory


# 创建模型

model = create_model(args.model, pretrained=args.pretrained)

# 创建 Timm 模型量化器

quantizer = ModelQuantizerFactory.get_model_quantizer('Timm', args.model, model, args.quant_config)

# 量化校准

quantizer.calibration(dataloader_train, args.batch_size, save_calib_model=True)

# 量化模型导出

quantizer.export_onnx(data_shape, onnx_path, dynamic_axes)
```

Timm 模型库使用 MTPQ 量化的例子

核心特性 #1: 自动化的量化算子插入

- **自动化 Q/DQ** 节点插入，内置 Module Converter 和 Pattern Matcher
- **Module Converter**
  - 转换常用模块为量化后模块， 并将不支持的算子转换为支持的等价算子
- **Pattern Matcher**
  - 使用 torch.fx 支持特定拓扑模式的匹配, 并根据 TensorRT 支持的图优化方式进行插入

核心特性 #2: 高性能的量化算法

- 集成了 SOTA 量化算法，能够获取更高的精度
- 支持的 PTQ 方法 (Naïve PTQ、AdaRound、支持自动化敏感度分析的 Partial PTQ)
- 支持的 QAT (Naïve QAT、LSQ / LSQ+、Stable-LSQ)
- 自研 Stable-LSQ ： 提升 LSQ 在零点附近的不稳定优化过程，从而达到更高量化精度

$$S(p_s) = \begin{cases} 0.5 * p_s^2, & |p_s| \leq 1 \\ |p_s| - 0.5, & p_s > 1 \end{cases}$$

$$\frac{\partial S}{\partial p_s} = \begin{cases} p_s, & |p_s| \leq 1 \\ -1, & p_s < -1 \\ 1, & p_s > 1 \end{cases}$$

ImageNet 性能对比

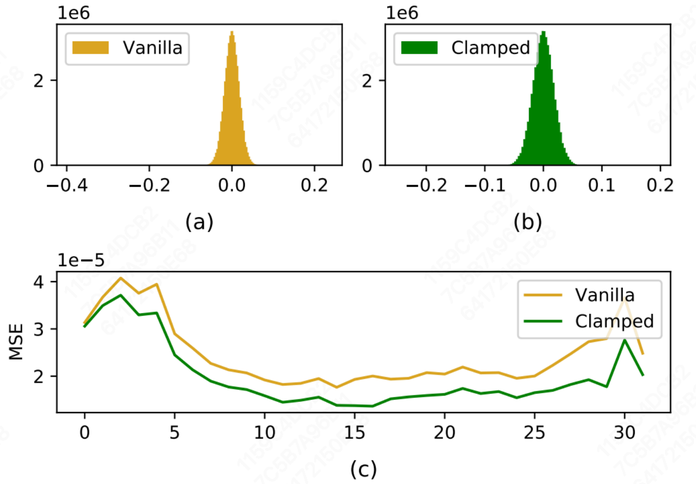
Model	FP32 (%)	LSQ INT4	Stable-LSQ INT4
ResNet18	69.758	69.518 (-0.240)	70.044 (+0.286)
ResNet50	76.130	75.958 (-0.172)	77.366 (+1.236)

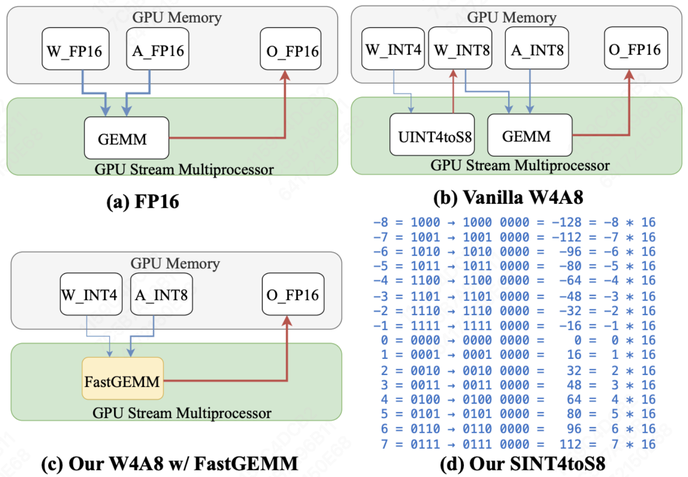
核心特性 #3: 可扩展性和框架支持

- 极简接口设计
  - 对用户代码的非侵入式设计
  - 几行代码完成量化功能
  - 支持自定义接口
- 常见代码框架支持
  - Timm
  - OpenMMLab (MMCls / MMSeg / MMSegmentation)
  - HuggingFace

核心特性 #4: 大语言模型量化

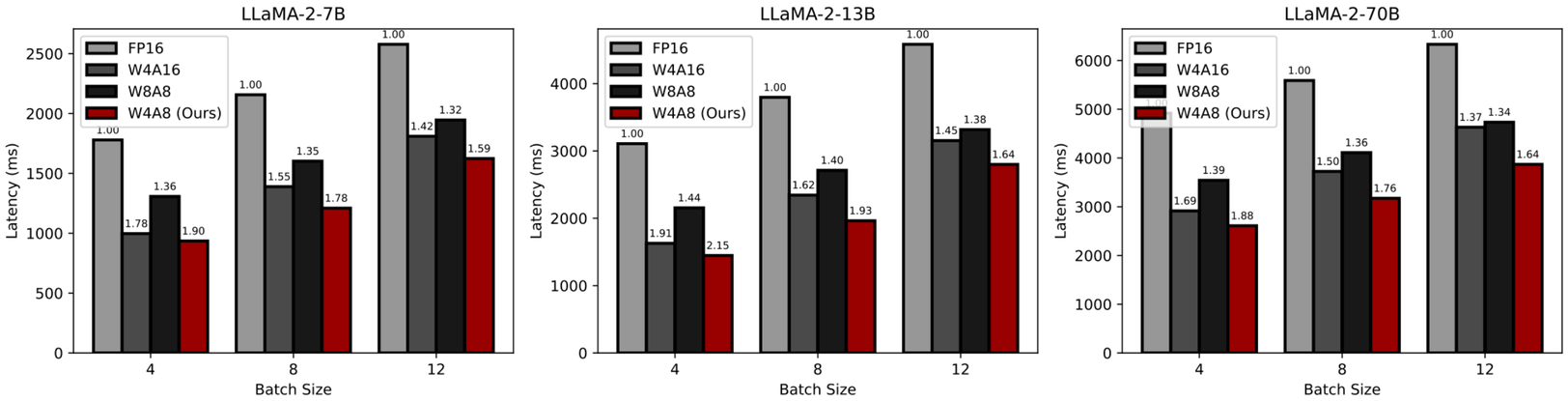
- 自研 LLM 量化算法 – OdysseyLLM
  - 逐通道对称量化
  - 可学习的权重裁剪 AWC
  - 量化权重补偿 GPTQ (interleaved with AWC)
- 融合 W4A8 Kernel 实现 – FastGEMM
  - Kernel 融合
  - SINT4toSINT8 转换
  - 减法指令消除



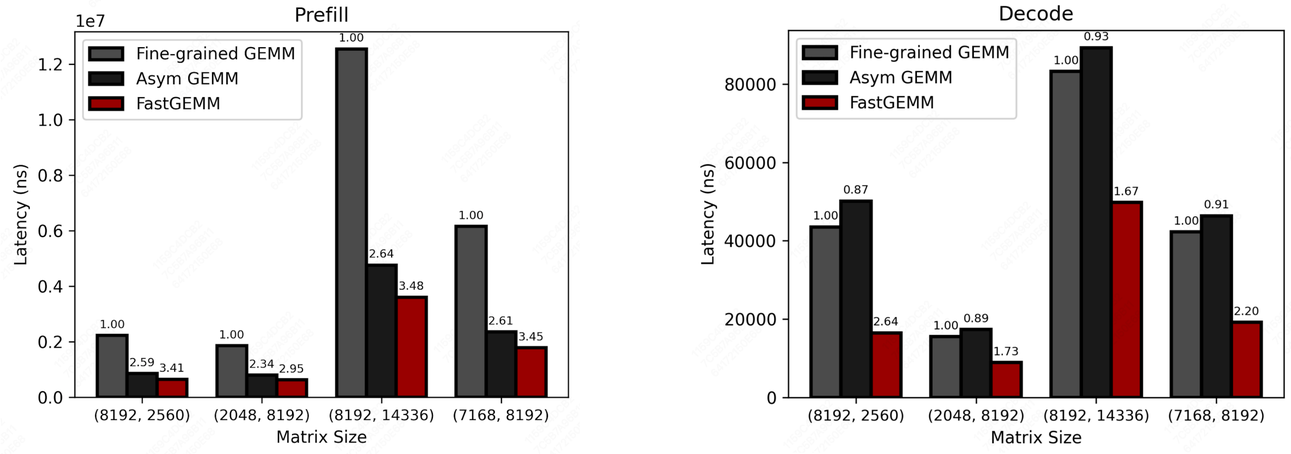


Li et al. A Speed Odyssey for Deployable Quantization of LLMs, https://arxiv.org/abs/2311.09550

核心特性 #4 (续.) LLM 推理性能测试



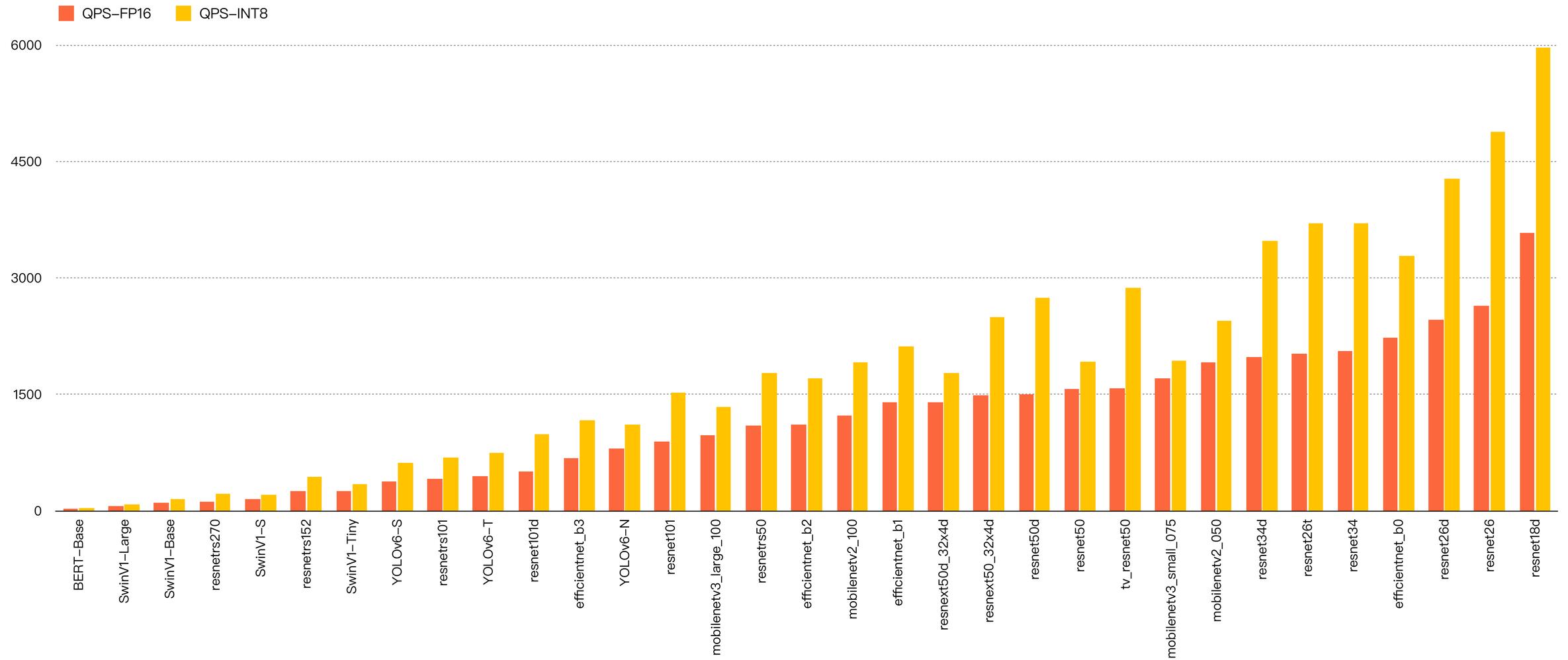
LLaMA-2 模型在 FP16、W4A16、W8A8、W4A8 精度下的性能对比



FastGEMM 与 Fine-grained GEMM 和 Asymmetric GEMM 的性能对比

性能基准评测：工业级模型加速

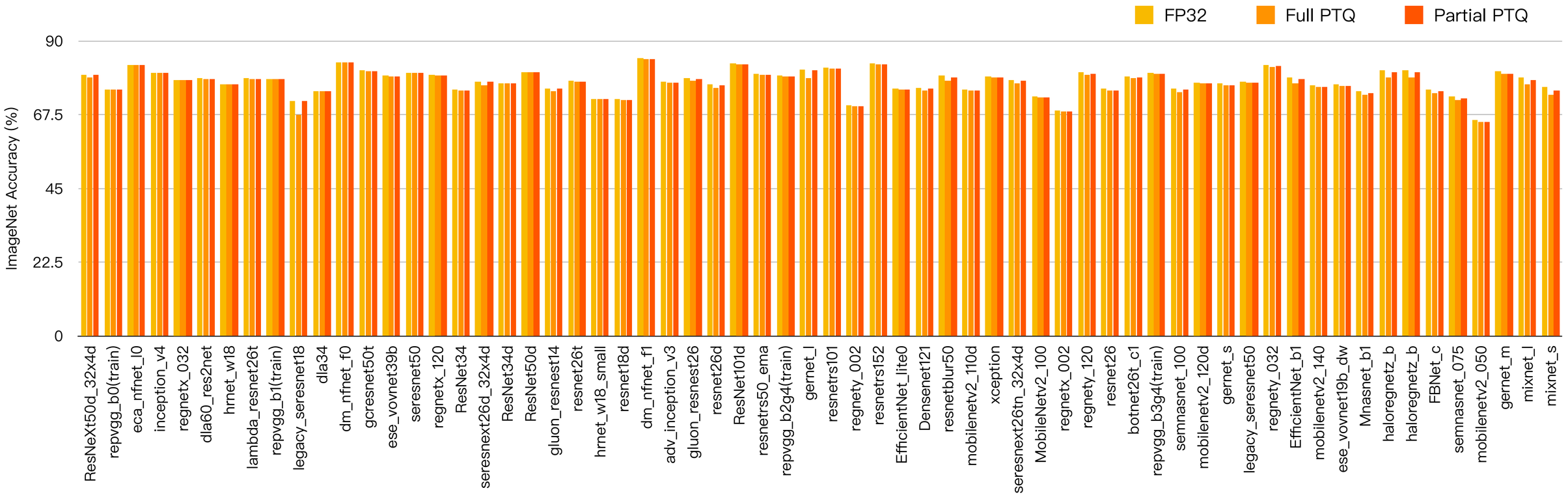
相比 TensorRT FP16，平均 58% QPS 提升



性能基准评测: ImageNet 模型精度对比

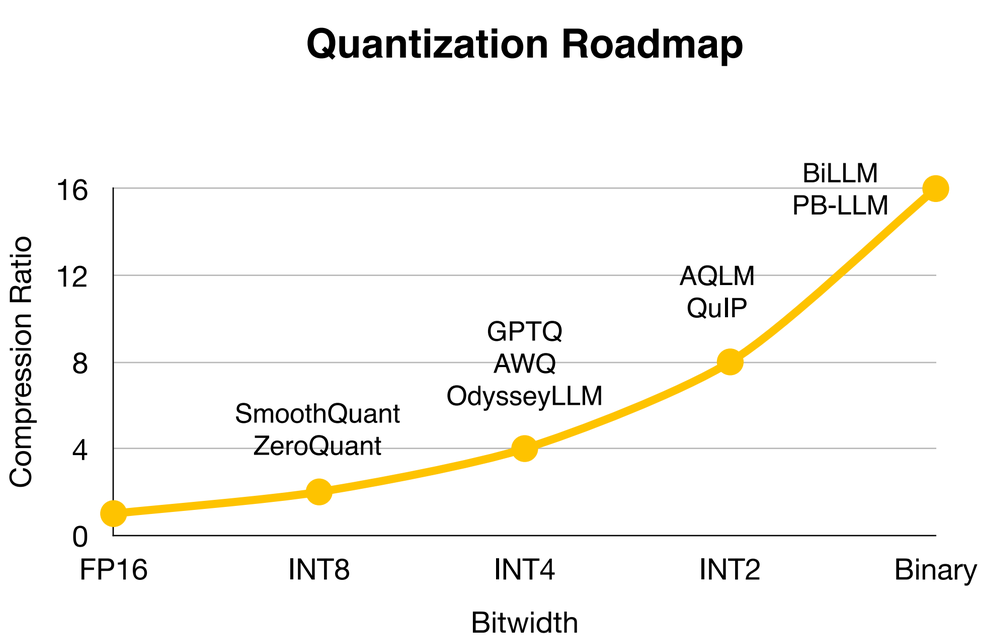
相比FP16， 平均仅 0.41% 精度损失

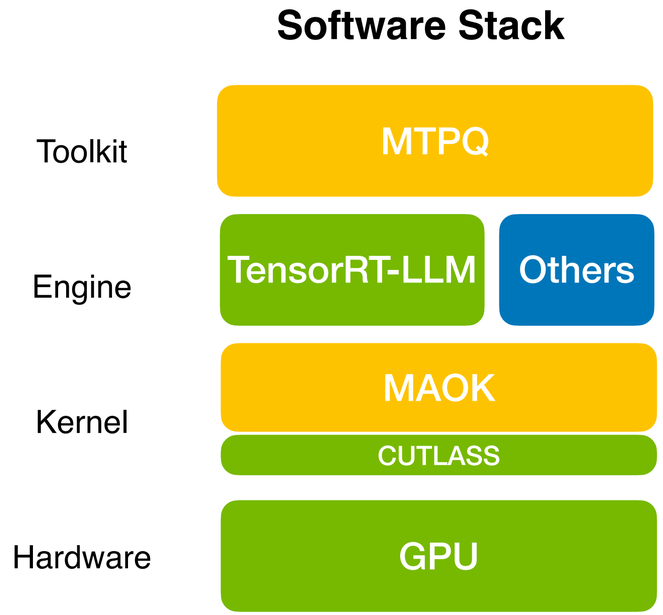
量化 PyTorch Image Models (Timm)



展望

支持低比特 LLM 量化，支持 TensorRT-LLM 部署





美团

谢谢