大模型计算-HW4

November 2025

1 Introduction

本次作业你需要对不同的量化方法进行模拟, 在 Qwen3-1.7B 模型上测试不同量化方法的精度表现以及在 nanovllm 中推理的吞吐表现。并通过 torchao 对模型使用 SmoothQuant 方法量化并测试推理速度 (prefilling).

2 Preliminaries

2.1 INT8/FP8 与细粒度量化

INT8 与 FP8 被广泛用于目前 LLM 的训练和推理中，可以说，量化到 8BIT 是目前 LLM 用于 serving 的基本配置。在 FP8、MXFP 被硬件支持之前，INT8/INT4 是为数不多用于低精度计算的数据格式，而由于均匀分布的 INT 与浮点数训练的 LLM 权重/激活值分布不同，直接应用 per-tensor 的 INT 量化会导致较大的性能损失。

由此，许多工作开始使用更加细粒度的量化，在不修改矩阵乘 kernel 的前提下，我们最多可以做到 per-token 的量化粒度。而在 Marlin、Jetfire 以及 DeepseekV3 后，在矩阵乘 K 维度的组量化已经成为目前低精度量化的标配。

对于 Deepseek V3 而言，尽管 FP8 实际上并不需要十分细粒度的量化，但由于 Nvidia GPU 的 FP8 TensorCore 在 Ada/Hopper 上有精度问题，Deepseek 仍然使用了 128 粒度的 K 维度量化。

2.2 TorchAO

TorchAO是一个 Torch 原生支持量化的仓库。目前已经在 A100, H100, B200 等显卡上支持了 int4, int8, float8 等多种数据格式的量化。安装 torchao 非常简单，通过如下命令即可：

```
pip install ao
```

使用 torchao 对 huggingface 模型进行量化需要定义 Config，目前支持的 Config 定义在 quant_api.py当中。以 Int8 WeightOnly 量化为例

```
from torchao.quantization import quantize_, Int8WeightOnlyConfig
quantize_(model, Int8WeightOnlyConfig())
```

量化的粒度（grannulity）可以通过传入参数来控制，不同的量化类型所支持的粒度不同，支持的类型可以在 quant_api.py 当中查看。以 Int8 WeightOnly 量化为例，如果不传入任何参数，会进行 per-channel 量化。除此之外，也可以进行更细粒度的分组量化，通过传入 group_size 自定义分组的大小：

```
from torchao.quantization import quantize_, Int8WeightOnlyConfig
quantize_(model, Int8WeightOnlyConfig(group_size=128))
```

类似的，如果希望进行 Weight 和 Activation 的 Int8 量化，需要选择不同的 Config:

```
from torchao.quantization import quantize_, \
    Int8DynamicActivationInt8WeightConfig
quantize_(model, Int8DynamicActivationInt8WeightConfig())
```

2.3 SmoothQuant

SmoothQuant 是一种用于大语言模型的 PTQ 方法，在不重新训练模型的情况下，实现对激活与权重的低比特量化。其核心思想是 **通过缩放因子的重新分配，将激活中的尖峰（outliers）平滑到权重中**，从而降低激活的动态范围，使其更容易被量化。在 Transformer 的线性层计算中，我们有

$$y = xW,$$

其中 x 为激活，W 为权重。若激活部分某些通道存在极端大值，直接对 x 进行低比特量化会导致量化误差极大。

SmoothQuant 引入一个按通道的缩放因子 s，并利用如下数学等价变换：

$$xW = \left(\frac{x}{s}\right)(Ws),$$

其中 s 是一个可学习或通过统计获得的正数缩放向量。该变换不改变层的输出，但会将激活 x 的动态范围缩小，从而大幅降低激活量化的难度；同时，权重 Ws 的变化可以通过离线量化处理，误差可控。

SmoothQuant 无需重新训练即可使用（Zero-shot PTQ），而且显著减少了激活量化误差。在 TorchAO 当中， smoothquant.py已经基本实现了 SmoothQuant 的逻辑。

3 Task

3.1 量化精度测试

基于修改后的 nanovllm 代码，利用并修改 utils/quantization.py，测试不同量化方法在下游任务中的精度。

仓库提供了 test_mmlu.py 与 test_ppl.py 测试 mmlu 上 5 shots 的 accuracy 以及 wikitext 上的 ppl. config 中添加了 linear_dtype 和 weight_quant_fn，可以选择性使用。

- 测试 weight only 的 INT8/FP8 per-tensor 和 per-row(per-channel) 量化对精度以及 ppl 的影响，是否精度越差结果越差?
- 测试 weight only 的 INT8/FP8 不同 group size 的组量化精度，INT8 和 FP8 对组大小的要求相同吗?
- 简述 per-tensor / per-row / per-group 量化在系统实现中的影响，量化/反量化是否可以 fuse 到已有的 kernel 中?

3.2 量化吞吐测试

基于修改后的 nanovllm 代码，利用并修改 utils/quantization.py，测试不同量化方法在下游任务中的速度。

- 在 RTX4090 上测试 INT8/FP8 per-row quantization 在实际推理中的吞吐，对 prefilling 和 decoding 各有什么影响?

- 将低精度矩阵乘修改为伪量化乘法，RTX4090 上 int8 与 fp8 的精度是否相同？为什么？

3.3 TorchAO

- 参考 smoothquant.py 实现对 Qwen2.5-1.5B 的 Int8 格式的 Weight 和 Activation 的量化脚本。
- 实现 FP8 格式的 Weight 和 Activation 的 SmoothQuant 量化脚本。
- 比较 FP8 和 Int8 两种格式量化的模型精度，可以自主选择评价指标。

4 DDL

本次作业 12.31 截止. 请提交一份 PDF 报告.