1.显存问题

1. 大模型大概有多大，模型文件有多大?

大模型也分为**不同的规格**，一般模型的规格会体现在模型的名称上，例如 LLaMA2-13b，13b 就是其模型参数量的大小，意思是 130亿的参数量。大模型的文件大小与其参数量有关，通常大模型是以半精度存储的， Xb 的模型文件大概是 2X GB多一些，例如 13b 的模型文件大小大约是 27GB 左右。

2. 能否用4 * v100 32G训练vicuna 65b?

一般来说**推理模型需要的显存约等于模型文件大小，全参训练需要的显存约为推理所需显存的三倍到四倍**，正常来说，在不量化的情况下4张 v100 显卡推理 65b 的模型都会有一些吃力，无法进行训练，需要通过 **LoRA 或者**QLoRA** 采用低秩分解的方式才可以训练。

3.如何评估你的显卡利用率?

这些方法主要是用来评估 **GPU 利用率** 的指标，以便了解在深度学习模型训练过程中，GPU 资源是否得到了充分的利用。通过这些计算方式，可以帮助开发者优化模型训练过程，提升性能。下面分别介绍每种方法。

1.FLOPS比值法

FLOPS（浮点运算每秒）是衡量计算设备（特别是GPU）性能的一个重要指标。FLOPS比值法通过计算实际运行时的 **FLOPS** 与显卡理论上的 **峰值FLOPS** 的比值，来估算 **GPU利用率**。

公式：gpu利用率 = 实测的FLOPS / 显卡理论上的峰值FLOPS

步骤：

1. **实测FLOPS**：在实际的训练过程中，可以通过 **DeepSpeed** 等工具记录模型训练的实际 FLOPS。假设 **DeepSpeed** 显示实测 FLOPS 为 **100T FLOPS**。
2. **理论峰值FLOPS**：这是显卡在理论上能够达到的最大计算性能。比如， **NVIDIA A100** 显卡的理论峰值为 **312T FLOPS**。
3. **计算GPU利用率**：将实际FLOPS除以显卡的理论峰值FLOPS，得出GPU的利用率。

示例：

如果实测FLOPS为100T FLOPS，而显卡理论峰值为312T FLOPS，则计算：

```
gpu利用率 = 100T FLOPS / 312T FLOPS ≈ 32.05%
```

这意味着GPU的计算能力在训练过程中仅得到了约32.05%的利用。

优缺点：

- **优点**：简单直观，可以快速评估GPU的计算利用情况。
- **缺点**：实际训练中的性能瓶颈可能不仅限于计算资源，还包括内存、带宽等因素，因此这种方法可能会高估或低估GPU利用率。

2.吞吐量估计法（Throughput Estimation）

吞吐量估计法通过 **训练时的吞吐量** 来评估 GPU 利用率。吞吐量通常是指每秒钟能处理多少个样本或者 token（对于语言模型而言）。

公式：

```
吞吐量 = example数量 / 秒 / GPU * max_length
gpu利用率 = 实际吞吐量 / 论文中的吞吐量（假设利用率100%）
```

步骤：

1. **计算实际吞吐量**：假设在训练时，处理速度为每秒 3 个样本，使用 4 张卡，总的吞吐量为：

   ```
   吞吐量 = 3 examples/s * 4 GPUs * 2048 max_length = 1536 token/s/GPU
   ```

2. **获取论文中的吞吐量**：根据相关论文（如 **LLaMA** 的论文），你可以找到模型训练时的理论吞吐量，假设 LLaMA 论文中显示 **7B 模型的吞吐量为 3300 token/s/GPU**。

3. **计算GPU利用率**：通过实际吞吐量与论文中的吞吐量比值来估算 GPU 的利用率：

   ```
   gpu利用率 = 1536 token/s / 3300 token/s ≈ 46.54%
   ```

优缺点：

- **优点**：这种方法可以很容易地与论文中的已知数据进行比较，尤其是在训练超大模型时。
- **缺点**：它依赖于你能够获取到论文中的吞吐量数据，且吞吐量受多个因素影响（如 batch size、学习率、硬件差异等）。

3.Torch Profiler分析法

Torch Profiler 是一个用于分析 PyTorch 训练过程的工具，它能够记录每个操作、函数以及 GPU 上的各个计算模块的执行时间。通过 **TensorBoard** 等工具展示这些结果，可以帮助我们深入理解模型训练过程中每个阶段的 GPU 利用情况。

步骤：

1. **使用Torch Profiler**：通过在训练代码中集成 **torch.profiler**，可以获得详细的性能分析数据，包括每个计算操作的时间。
   - 记录每个操作的 **执行时间** 和 **计算时间**。
   - 生成每个 GPU 核心的利用率数据，如 Tensor Core 的利用率。
2. **展示在TensorBoard上**：将 profiler 生成的数据可视化到 **TensorBoard**，在 **GPU kernel视图** 下查看 GPU 核心的利用情况。
   - **Tensor Core利用率**：例如，记录到某个操作的 **Tensor Core利用率** 可能为 **30%**，表示该操作只使用了GPU计算单元的 30% 能力。

优缺点：

- **优点**：这种方法非常细致，可以准确分析每个 GPU 核心的利用情况，帮助开发者发现训练中的性能瓶颈。
- **缺点**：需要额外的分析工作，且可能对训练过程产生一定的性能开销。

4. 如何查看多机训练时的网速?

```
iftop -i eth2 -n  -P
```

`iftop`是外置的命令，可以监控发送流量，接收流量，总流量，运行`iftop`到目前时间的总流量，流量峰值，过去 2s 10s 40s 的平均流量。

5. 如何查看服务器上的多卡之间的NVLINK topo?

```
nvidia-smi topo -m
```

6. 如何查看服务器上显卡的具体型号?

```
cd /usr/local/cuda/samples/1_Utilities/deviceQuery
make
./deviceQuery
```

7. 如何查看训练时的 flops? （也就是每秒的计算量）

如果基于deepspeed训练，可以通过配置文件很方便地测试。

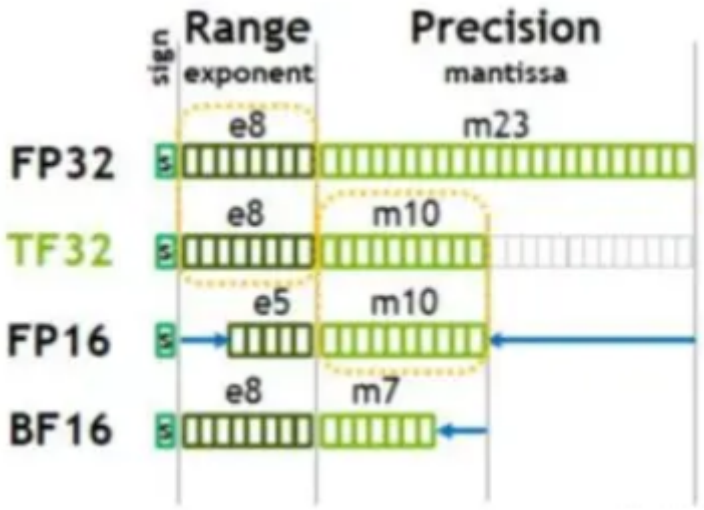
```
{
  "flops_profiler": {
    "enabled": true,
    "profile_step": 1,
    "module_depth": -1,
    "top_modules": 1,
    "detailed": true,
    "output_file": null
    }
}
```

8. 如何查看对 deepspeed 的环境配置是否正确?

```
ds_report
```

9. TF32 格式有多长?

TF32（TensorFloat32）是 NVIDIA 在 Ampere 架构推出的时候面世的，现已成为 Tensorflow 和 Pytorch 框架中默认的32位格式。用于近似 FP32 精度下任务的专有格式，实际上约等于 FP19 也就是19位。



FP32（32-bit floating point）：通常使用 **1位符号位 + 8位指数位 + 23位尾数（Mantissa）**。它提供较高的数值精度和较大的动态范围，适用于要求高精度计算的任务（如科学计算、数值仿真等）。

TF32：使用 **1位符号位 + 8位指数位 + 10位尾数（Mantissa）**。这意味着 TF32 的精度比 FP32 更低，但它通过降低尾数位的位数来提高计算效率。实际而言，TF32 近似等于 **FP19**，即尾数部分只有 19 位。