第四篇 以Nvidia为例，如何在Perf中添加标准训练case

FlagPerf立项以来，已累计添加20+个模型，未来持续在质量上提升，数量、场景上拓展，包括不限于大模型和推理case的添加，本篇文章，咱们就从Nvidia实例出发，带大家加case～

1. 标准Case定义

我们先复习一下标准case的定义：

“ 如今，业内主流硬件厂商以英伟达为首，我们将标准case定义为模型-训练框架-英伟达(以下简称**case)**的一个组合。 同时，结合各家芯片公司软件栈的设计和市场选择偏向，我们选择Pytorch作为主要训练框架。因此，标准case多以“Model+Pytorch+Nvidia”的组合为主。

标准case以Nvidia GPU作为运行参照，代码实现上面依赖Cuda，**原则上**不依赖其它特定芯片厂商的软件包，如NVIDIA/apex相关实现不会出现在标准。如出现如apex类的优化包，则应当放在nvidia目录下。”

2. 标准Case代码

2.1 代码结构

标准Case实现路径在training/benchmarks/model/framework/下Nvidia基础版本实现，标准Case代码以run_pretraining.py脚本为入口，该脚本由start_<framework>_task.py脚本在容器内调用执行。整个代码建议遵循常见的训练pipeline，以glm-pytorch case为例，包括以下几部分：

```
├── glm
│   └── pytorch
│       ├── config   # 【必选】case基本配置目录
│       │   ├── __init__.py
│       │   ├── _base.py  # case基础配置：包含了case运行的所有配置参数
│       │   └── mutable_params.py # 可变参数列表：定义了厂商可以覆盖的配置项名称。
│       ├── dataloaders # dataloaders定义构建dataset, dataloader相关逻辑（train &
│       │   ├── __init__.py
│       │   └── dataloader.py  # dataloader: 定义dataset, dataloader的构建逻辑，
│       ├── model    # 模型定义
│       │   ├── __init__.py
│       │   ├── layers
│       │   └── models
│       ├── optimizers # 优化器相关
│       │   ├── __init__.py
```

```
16 │       │   ├── fp16_optimizer.py # fp16优化器
17 │       │   └── loss_scaler.py    # loss缩放，解决FP16类型数据下溢问题。
18 │       ├── readme.md  # 【必选】case readme文档
19 │       ├── run_pretraining.py  # 【必选】执行训练入口脚本，可以使用dev/docs/run_pr
20 │       ├── schedulers  # lr schedulers
21 │       │   ├── __init__.py
22 │       │   └── base.py
23 │       └── train
24 │           ├── __init__.py
25 │           ├── evaluator.py  #  evaluator: 定义验证evaluate方法
26 │           ├── trainer.py    # trainer:  定义训练pipeline
27 │           ├── trainer_adapter.py # trainer_adapter: 定义训练流程的各个步骤的具体
28 │           └── training_state.py  # training_state: 保存训练状态的dataclass
```

2.2 添加流程

1. 数据集和checkpoint(如有)准备
2. 从头开始训练，保存ckpt(可选), 验证原始仓库精度达标，结果作为target acc
3. 从原始仓库分离模型、优化器、学习率调度器等关键训练组件
4. 整理trainer训练脚本，依据配置文件开发trainer_adapter模型适配器
5. 编写run_pretraining入口函数，串联各训练组件、训练脚本与Perf框架相应功能
6. 撰写nvidia-1x8 config，1x8 作为经典配置，优先支持
7. 在Perf框架下测试1x8全流程（结果应与2相同，保证收敛）
8. 测试1x1, 2x8配置
9. 补充case文档，模型文档
10. 对照PR提交规范，提交PR

2.3 关键代码

在training/benchmarks下添加<model/framework>子目录，该目录下为一个新增case的主题代码

```
1 .
2 ├── config      # 【必选】case基本配置目录
3 ├── dataloaders
4 ├── model
5 ├── optimizers
6 ├── readme.md   # 【必选】case文档，规范参考 https://qgsq1vkxhz.feishu.cn/docx/NMA
7 ├── run_pretraining.py #【必选】执行训练入口脚本，可以使用dev/docs/run_pretaining.p
8 ├── schedulers
9 └── train
```

原则上，标准实现中不放特有优化，该部分的空间留给各个厂商去拓展，如下面所示，在training/<vendor>/<model-framework>目录下存放厂商特有的实现，如Nvidia下的apex优化。

```
training/nvidia/glm-pytorch/
├── config # nv下的配置文件
└── extern # nv上的特有实现，如apex优化
```

3. 添加方式

3.1 容器内训练

3.1.1 代码添加

我们在项目中给了一个参考入口文件，training/benchmarks/model/framework/run_pretraining.py.example

开发者根据该脚本中的标记TODO的位置进行修改，串接整个训练pipeline。

为了降低大家添加模型的难度和门槛，我们只要求保证关键接口不变的情况下，可接受自定义内部实现。

另外，使用Pytorch框架的各个领域的训练任务也会存在些许区别，所以我们也接受训练pipeline的调整，只需要整体逻辑尽可能一致，不要去完全一致。

这部分链接到文件模块trainer、trainer_adapter、config, 都属于标准case的必须项，开发者可参考仓库内已有的case，比如mobilenetv2、faster rcnn 来撰写。

一般我们推荐优先支持的是1x8的实验配置, _base.py作为config模块的基石，存放和硬件厂商无关的基本配置，如模型结构，基础训练超参设置，凡和硬件厂商有关的配置，放置于厂商config目录下，不同层级的配置优先级为：

test_conf.py > config_A100x1x8.py > _base.py，三者都存在的情况下，覆盖式生成最终版本。

提交的最终版本代码，期望训练时间在一周内能够在Nvidia上精度达标。目前以一周来区分，如果模型训练时间过久，则 case by case的讨论对策。

3.1.2 添加实验配置

- 文件路径：training/nvidia/<model-framework>/config
- 文件内容：不同配置下的模型超参数及硬件依赖参数
- 配置文件列表如下：（以GLM-Pytorch为例）

```
training/nvidia/glm-pytorch/config
├── config_A100x1x1.py      # 必选，单机单卡配置，性能结果和精度验证
```

```
3 ├── config_A100x1x8.py       # 必选，单机8卡配置，性能结果和精度验证
4 ├── config_A100x2x8.py       # 必选，2机8卡配置，性能结果和精度验证
5 ├── environment_variables.sh # 环境变量文件，在容器启动后，运行case之前会source该文件
6 └── requirements.txt         # 容器中安装python依赖包。FlagPerf框架在启动容器后，在这
```

理论上，不同配置上的超参数应该有所不同，但主旨之一是需要保证尽可能高的显卡利用率。

以上工作完成，满足进入容器中启动训练任务的目标，下面的工作保证在容器外能以统一方式批量启动测例。

在上面我们提到，标准的配置优先级是test_conf.py(data dir) > config_A100x1x8.py > _base.py，该角度是从用户使用视角来说明，从文件本身的内容视角，标准Case实现的配置又可以分为两大类：

基本配置

基本配置是模型训练及运行环境相关的配置参数，主要分为两大类: 模型超参(如LR)和训练配置(log_freq等)，路径为training/benchmarks/<模型>/<框架>/config/_base.py和mutable_params.py, 其中mutable_params.py 中定义的参数表示可覆盖参数项

- 路径：<model>-<framework>/config/_base.py 。**定义模型训练相关的所有参数，及case运行环境需要的基本参数**。模型名称、模型结构、数据集、checkpoint、超参数、分布式参数等。
- 配置项说明如下，可参照[standard-case-config-base.py.example]：
- 必选参数：
  - `vendor`，值为"nvidia"即可，会在运行FlagPerf时被配置在test_conf里的vendor值覆盖。
  - `data_dir`，值为""空即可，会在运行FlagPerf时被配置在test_conf中对应case配置项data_dir_container覆盖。
- 可选参数：
  - `train_data`：训练数据路径，填写相对 `data_dir` 的路径
  - `eval_data`：评估数据路径，填写相对 `data_dir` 的路径
  - `init_checkpoint`：初始化模型checkpoint文件，填写相对 `data_dir` 的路径，一般提供文件名即可，下载checkpoint文件后放置于 `data_dir`
- 等等

Nvidia适配的配置

Nvidia适配的配置是指标准Case运行在Nvidia GPU环境的配置参数。作为Nvidia的标准配置，可以和基础配置一致，也可以在运行时覆盖标准Case的基础配置参数。在Nvidia GPU上运行所需的配置文件放在training/nvidia/<model>-<framework>/config目录下，可以看作是Nvidia适配标准Case的配置项，由于训练规模不同，可以给出多个配置文件。

在FlagPerf运行时，会根据test_conf里的case配置项选择加载哪个配置文件。配置文件命名为：config_<machine>x<nnodes>x<nproc>.py，例如单机4卡的A100环境运行，使用

config_A100x1x4.py，这里主要放置是厂商适配case时可覆盖的参数，一般定义在自己设备上跑该Case最优的配置。

3.2 容器外启动

1. 在training/run_benchmarks/config/test_conf.py中添加新标准case的key-value, 验证以run.py方式启动有效性。

以GLM和Bert为例,

```
# Set the case dict you want to run here.
'''
# Users must use {
    "model:framework:hardwareID:nnodes:nproc:repeat": "dataset path"}
'''
CASES = {
    "bert:pytorch:A100:1:8:1": "/home/datasets_ckpt/bert/train/",
    "glm:pytorch:A100:1:8:1": "/home/datasets_ckpt/glm/train/",
}
```

2. 在training/run_benchmarks/config/cluster_conf.py中添加标准case运行机器

```
# Hosts to run the benchmark. Each item is an IP address or a hostname.
HOSTS = ["10.209.20.12","10.209.20.13"]
```

3. 【必需】在run.py 脚本启动完整测试

```
python3 ./run_benchmarks/run.py
```

4. 在training/nvidia/docker_image/pytorch/目录下撰写环境依赖包

注：调试运行时可在容器中进行，最终提交前需run.py 完整测试，验证代码工作正常。关于配置方法，可参考：https://github.com/FlagOpen/FlagPerf/blob/main/training/README.md

4. 验证达标要求

- 以Perf方式启动训练模型收敛达到目标精度(NV上原始代码单机8卡精度值)
- 单个case的收敛时间在1周内
- 多机/多卡吞吐量加速比符合预期，需提供1*1，1*8和2*8三种配置验证单卡，多卡，多机场景下的性能数据

- 提供训练过程和benchmark结果日志(NV)，包括训练中的N个steps的和最终结果输出。
- **提交模型、case、厂商三类文档，具体介绍见上篇。**

文档模版： [模型README文件模版](../readme-templates/model-readme-template.md) 和 [case README文件模版](../readme-templates/case-readme-template.md)。

以上就是在FlagPerf中添加一个标准case的流程啦，欢迎大家一起来给Perf添砖加瓦～