第三篇 FlagPerf中标准case的定义和添加&适配原则

FlagPerf发展至今，在前进过程中沉淀下多个“标准”。所谓“不以规矩，不成方圆”，我们期望在各项标准的规范下，FlagPerf成为一个可靠的芯片评测平台。

1. 标准Case定义

如今，业内芯片厂商百花齐放，主流硬件厂商以英伟达为首，因此我们将标准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。

对于不同类型的芯片，芯片厂商根据芯片的实际情况支持/扩展标准Case的模块和接口，完成适配。

目前，FlagPerf中模型的标准case可支持多家芯片厂商，如英伟达、天数、昆仑芯，部分模型支持多训练框架，如resnet50、Bert等。

2. 标准Case的模型选择

FlagPerf旨在和大家共同探索开源、开放、灵活、公正、客观的AI芯片评测体系，建立评测软件生态，提供行业价值，因此在模型选择上面考虑以下几个方面：

a. 尽量覆盖典型应用场景典型模型

b. 及时跟进新的热门的模型，便于用户及时评测

c. 模型代码标准实现源自github公开高认可的仓库

3. 标准Case代码标准

在添加标准case的过程中，我们发现很多细节问题如果没有明确的定义，在执行过程中处于模糊边界容易给后面的验证带来不确定性，当不确定性累计过多的时候，case的可靠性可能就会大打折扣。基于此，标准case的添加标准在一砖一瓦中慢慢形成。

接下来，给大家分享下目前关于标准模型的标准定义，当然，这些标准还在不断完善更新中，欢迎大家提出宝贵的建设性意见。

下面我们从代码结构，添加原则，文件规范、适配原则四个角度介绍标准case的“标准”定义和适配规范。下一篇文章，我们将以NV典型模型为例，再给大家介绍一个标准case如何添加到Perf项目，看看从标准制定到落实添加要经历哪些环节。

3.1 代码结构

标准Case实现路径在training/benchmarks/model/framework/下，仅为NVIDIA上基础版本实现，厂商负责适配标准实现下的各家芯片版本，若是厂商需要额外拓展支持的算子/接口则放在【厂商/extern】模块下进行。

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

3.2 添加规范

- 总结流程如下

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

以下为详细解释。

3.1 准备工作

- 环境准备

  优先使用Perf已有的镜像版本，未来Perf会支持2-3个版本镜像以供选择，缺包可基于已有镜像新增安装包。

- 原始代码训练验证

  原始代码的验证是非常必要的。有时候我们会碰到原始参考不收敛，甚至有bug的情况，这时候能够及早的发现问题非常重要。

  在确定待添加的模型代码链接后，我们建议使用**原始代码**的数据集和代码配置进行**单机8卡**模型复现。作用如下：

  - 确保源代码没有质量问题避免无用功
  - 确定目标精度，若NV的精度在合理值，则该值作为其他厂商对标精度，且作为配置文件中target_acc的值

- 数据集记录
  - 将数据集下载路径写入文档，便于复现结果。若原始代码不带数据集，使用业内公开知名数据集。其他必要的步骤，都可以写入文档，方便其他厂商/用户复现。

验证原始代码没有问题后，开始Perf的适配工作。

3.2 关键代码逻辑

在training/benchmarks下添加model/framework子目录，

```
.
├── config      # 【必选】case基本配置目录
```

```
├── dataloaders
├── model
├── optimizers
├── readme.md    # 【必选】case文档，规范参考 https://qgsq1vkxhz.feishu.cn/docx/NMA
├── run_pretraining.py #【必选】执行训练入口脚本，可以使用dev/docs/run_pretaining.py
├── schedulers
└── train
```

原则上，在training/nvidia/模型-框架目录下存放厂商适配部分，如nvidia下则存放NV上特有的优化。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。

```
.
├── config # nv下的配置文件
└── extern # nv上的特有实现，如apex优化
```

本着讲“原则方法”，不讲细节的初衷，这里只强调实现方式。

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能过NV上精度达标。关于训练时间定为一周，还有个小故事。早前，我们期望通过checkpoint的方式控制一个case能够在2-5h内训练达标。随着模型覆盖领域的拓展，我们发现，控制几个小时的收敛时间，并不能保证模型的收敛质量，比如大模型场景，训练几个小时并没有参考意义。

因此，我们及时调整原则，目前以训练时间一周来区分，如果模型训练时间过久，则 case by case的讨论对策。另一个角度来看，从头开始训练，也排除了checkpoint 在硬件厂商适配过程中变量因素，更加保证了模型的准确性。

3.3 训练实验配置

- 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）

```
├── config_A100x1x1.py       # 必选，单机单卡配置，性能结果和精度验证
├── config_A100x1x8.py       # 必选，单机8卡配置，性能结果和精度验证
├── config_A100x2x8.py       # 必选，2机8卡配置，性能结果和精度验证
├── 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
└── requirements.txt          # 容器中安装python依赖包。FlagPerf框架在启动容器后，在
```

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(lr等)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
- 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。

由于FlagPerf的一些代码设定，对配置文件路径和内容有一定要求。

基本配置

- 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](../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

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_model>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。

3.4 测例入口配置

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/目录下撰写环境必须包

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#readme

3.5 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。finished_info包括不限于：e2e_time、training_sequences_per_second、 converged、final_accuracy、raw_train_time、init_time等

4. 文档标准

我们对FlagPerf中的文档有三种区分：

1. 厂商文档，首次合作的厂商，需要提供文档向用户介绍Perf中模型适配芯片的基本情况
2. 模型文档，当模型作为标准case首次添加进Perf项目时，需要撰写文档介绍模型的基本情况
3. case文档，模型-框架-厂商作为一个case组合，在完成适配后需要提供文档介绍模型在该芯片上性能数据

以上三个文档，1&2只需要提供一次即可，3对于同一模型则根据适配厂商数量需要提供多次。

文档模版可参考： [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)。

5. 厂商适配原则

以上主要是在介绍标准case添加时候的原则，在完成标准case的添加后，不同的厂商会参与进来进行模型适配，最终一个模型可以在不同的芯片上运行，以此来达到芯片的评测的最终目的。

1. 基本原则

厂商基于标准case做硬件上的适配，适配原则：

1. 优先默认使用标准case实现，不做上层接口变动，理想情况下，只有底层算子的区别，对用户不感知；
2. 接受对模型做合理优化以此来提升模型的性能表现，如bs调整等，建议底层优化，暂不接受torch接口层优化, 具体可case by case讨论。
3. 对于标准case中厂商不支持的算子，可有合理替代方案，**具体请与FlagPerf研发团队联系。**

- 目前可扩展接口没有做严格限制，但原则上**为了保持训练workload基本一致**，从对benchmark的基本认知出发，需控制变量，要求适配过程中包括但不限于：模型结构和大小、优化器类型、数据集等内容**不能改变**。

2. 环境构建原则

我们遵循一个框架 + 一个厂商支持有限镜像的原则，因此，目前不同厂商仅需提供一个基础镜像支持所有模型，如有必要，也可支持多镜像。

因此，厂商在初次适配时，需要提供构建容器镜像的Dockerfile和脚本。如果framework_install.sh脚本中需要额外下载和安装软件包，建议使用packages目录存放，并在README文档中说明。

FlagPerf会根据这里的Dockerfile及脚本自动构建容器镜像，镜像名为：flagperf-vendor-framework, tag为t_vversion。出于性能考虑，FlagPerf不会在每次启动测试时构建新镜像，除非测试环境主机上不存在对应名称和tag的容器镜像。

3. 硬件监控脚本

FlagPerf启动Case运行的容器前，会启动系统监控信息的采集程序，在测试结束后，会结束系统监控信息的采集程序。主机CPU和内存的使用情况由FlagPerf自带的training/utils/sys_monitor.py采集。厂商需要提供自身芯片的监控信息采集脚本，放在training/vendor/目录下。

4. 厂商代码修改原则

4.1 厂商运行case需修改的参数

当模型层的支持都已完成，在运行case层面还需要配置以下参数

- cluster配置: training/run_benchmarks/config/cluster_conf.py 修改为厂商硬件IP

```
'''Cluster configs'''

# Hosts to run the benchmark. Each item is an IP address or a hostname.
HOSTS = ["10.1.2.2", "10.1.2.3", "10.1.2.4"]  # 设置集群节点ip列表

# ssh connection port
SSH_PORT = "22"
```

- case配置：training/run_benchmarks/config/test_conf.py，针对厂商软件栈情况配置启动容器相关参数

```
# Set accelerator's vendor name, e.g. iluvatar, cambricon and kunlun.
# We will run benchmarks in training/&lt;vendor&gt;
VENDOR = "nvidia"    # 这里改成产商自己的名称，提交代码需复原

# Accelerator options for docker. TODO FIXME support more accelerators.
```

```
ACCE_CONTAINER_OPT = " --gpus all"    # 这里设置为空

# XXX_VISIBLE_DEVICE item name in env
# nvidia use CUDA_VISIBLE_DEVICE and cambricon MLU_VISIBLE_DEVICES
# CUDA_VISIBLE_DEVICES for nvidia，天数
# MLU_VISIBLE_DEVICES for 寒武纪
# XPU_VISIBLE_DEVICES for 昆仑芯
ACCE_VISIBLE_DEVICE_ENV_NAME = "CUDA_VISIBLE_DEVICES"
```

4.2 适配代码组织结构

- 代码放在training/vendor/model-framework;目录下，分三个目录组织：
  - config目录，存放厂商配置文件，模型在厂商机器上获得最佳性能的参数值。environment_variables.sh和requirements.txt文件用于FlagPerf在启动容器后构建环境。运行测试Case前，会先sourche environment_variables，然后使用pip安装requirements.txt中指定的包。
  - csrc目录，放算子源码和编译安装脚本setup.py。FlagPerf会在启动容器后，运行测试Case前，调用setup.py进行算子的编译和安装。
  - extern目录，模型适配的代码，标准case不支持/底层调优的相关代码放置于此。【该处会重点review】

5. 适配达标要求

- 遵循控制变量法则，仅做适配芯片的必要修改
- 调优配置至能体现厂商性能的最优配置
- 多机/多卡吞吐量加速比符合预期
- 以Perf方式训练1x1、1X8、2X8模型收敛达到目标精度(标准case中的target acc)
- 支持硬件监控指标采样（必选：时间戳、使用率、显存使用率，可选：功耗、温度等，建议都有）

6. 文档规范

按模版建议提供厂商文档和每个Case的性能文档(精度必选，速度可选)，文档描述需与代码实现相符合。

以上就是目前FlagPerf中的所有原则性标准啦，覆盖添加标准case和厂商适配case，希望大家多多提出宝贵建议，帮助Perf成长为更好的评测平台。