GTS-Engine

Release 0.1

IDEA-CCNL

Feb 21, 2023

CONTENTS:

CHAPTER

ONE

关于 GTS ENGINE

GTS 引擎（GTS-Engine）是一款开箱即用且性能强大的自然语言理解引擎，能够仅用小样本就能自动化生产 NLP 模型。它依托于封神榜开源体系的基础模型，并在下游进行了有监督预训练，同时集成了多种小样本学习技术，搭建了一个模型自动生产的流水线。作为 GTS Factory 的核心组件，集先进的预训练模型、端到端的模型训练和推理于一体，提供一站式 NLP 开发与服务，让开发者能够在私有化环境上高效地开发模型，搭建服务。

1.1 应用场景

GTS Engine 着眼于小样本任务，支持开发者根据自己的数据集定制化开发下游任务。

GTS Engine 包含两个训练引擎：乾坤鼎和八卦炉。**乾坤鼎**系列是以 1.3B 参数的大模型为底座，通过大模型结合多种小样本学习技术进行训练和推理的引擎。**八卦炉**系列是以 110M 参数的 base 模型为底座，融合大模型、数据增强、协同训练等方法进行训练和推理的引擎。GTS Engine 支持的任务类型：

我们会在后续的版本中持续迭代，将通用的 NLP 任务集成进 GTS Engine。

1.2 使用 GTS Engine 的理由

- 先进的预训练模型
- 强悍的小样本学习能力
- 灵活易用的开发接口，让无算法背景的用户也能轻松开发自己的模型

CHAPTER

TWO

快速开始

2.1 服务启动

下面以启动乾坤鼎作为例，来让您快速上手 GTS Engine 的使用。首先提供以下三种启动服务的方法：

- *pip 启动*
- *源码启动*
- *Docker 启动*

2.1.1 pip 启动

```
# 为了避免与原来环境下载包的版本不同导致冲突，可以新建环境
conda create -n gts_project python=3.8
# 环境激活
conda activate gts_project
# 下载 gts-engine 包
pip install gts-engine
```

选择适合适配显卡的 cuda 版本 torch 安装，torch 版本要求 1.11.0，下载地址

示例：NVIDIA 版本 `Driver Version: 450.80.02, CUDA Version: 11.0` 可直接下载

```
# 下载成功 cuda 版本 torch，安装
pip install torch-1.11.0+cu113-cp38-cp38-linux_x86_64.whl
```

需要新建名为 `pretrained`，`tasks` 的两个文件夹建议按需下载模型（请见下表）。如果不下载，训练时会自动从 `huggingface` 下载，由于模型文件较大，可能因为网络原因导致下载失败，从而导致模型训练失败

```
CUDA_VISIBLE_DEVICES=0 python gts_engine_service.py --task_dir tasks --pretrained_dir␣
↪pretrained --port 5201
```

2.1.2 源码启动

GTS-Engine 源码地址

```
# 下载源码
git clone https://github.com/IDEA-CCNL/GTS-Engine.git
cd GTS-Engine
# 下载包
python setup.py install
#cuda 版本 torch 安装   具体参考上述 pip 启动的建议
pip install torch-1.11.0+cu113-cp38-cp38-linux_x86_64.whl
# 将下载好的 Erlangshen-UniMC-MegatronBERT-1.3B-Chinese 模型文件放在 pretrained  具体参考上
述 pip 启动的建议
mkdir pretrained
mkdir tasks
cd gts_engine
# 参数介绍参考上述 pip 启动的建议
CUDA_VISIBLE_DEVICES=0 python gts_engine_service.py --task_dir tasks --pretrained_dir␣
↪pretrained --port 5201
```

如下所示，代码目录 gts_engine 和预训练模型目录 pretrained、任务目录 tasks 在同一级目录，在 pretrained 目录中存放预训练模型文件，在 tasks 中存放任务相关的数据和模型。

```
├─GTS-Engine
|    ├── gts_engine
     ├─pretrained
|        ├── Erlangshen-UniMC-MegatronBERT-1.3B-Chinese
     ├─tasks
|        ├── tnews
```

您也可以通过命令行脚本进行训练和推理，具体示例请看 `examples/text_classification`。

2.1.3 Docker 启动

Docker 入门

```
#docker 下载
sudo docker push gtsfactory/gts-engine:v0
#docker 启动
#--mount 注: 目录挂载 source 对应的必须是存在的本地绝对路径
#-p 本地端口与 docker 端口映射
sudo docker run -it --name gts_engine \
-p 5201:5201 \
--mount type=bind,source=/usr/tasks,target=/workspace/GTS-Engine/tasks \
```

(continues on next page)

(continued from previous page)

```
gtsfactory/gts-engine:v0
# 更新代码
cd GTS-Engine
git pull
cd gts_engine
# 启动服务
CUDA_VISIBLE_DEVICES=0 python gts_engine_service.py --port 5201
```

2.2 API 接口使用

我们支持两种方式来使用我们的引擎：通过 Python SDK 的方式和 Web 页面的方式。通过 Python SDK 的方式请参考GTS-Engine-Client。

下面通过 fewCLUE 中的 tnews 任务为例，介绍 Web 页面使用引擎的方式。

- 1.*api* 调试界面
- 2. 创建任务
- 3. 查看任务列表
- 4. 查看任务状态
- 5. 删除任务
- 6. 上传数据
- 7. 启动训练
- 8. 停止训练
- 9. 开启推理模式
- 10. 推理
- 11. 结束推理

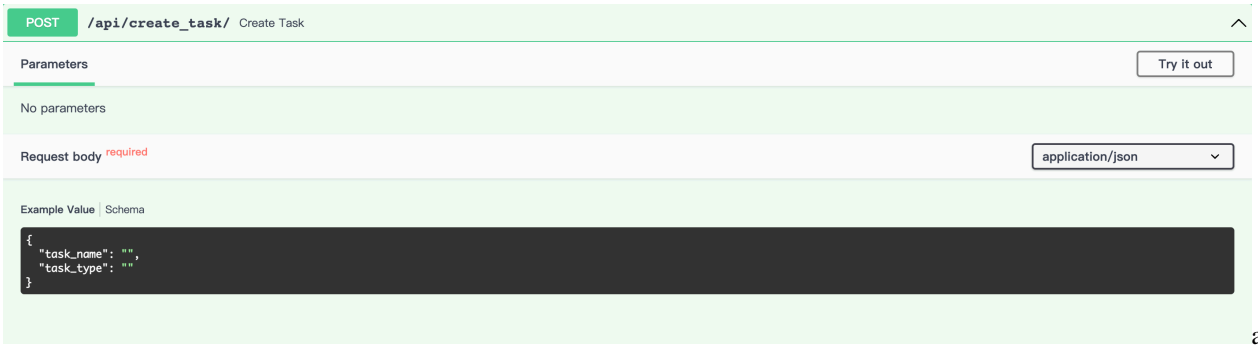
2.2.1 api 调试界面

GTS-Engine 启动成功后的会显示如下界面：

```
INFO:     Started server process [3509632]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:5201 (Press CTRL+C to quit)
```

用浏览器访问：http://0.0.0.0:5201/docs，跳转到 api 交互式界面：

2.2.2 创建任务



初始界面如下：

点击 `Try it out` 按钮，设置 task_name 和 task_type 对应的参数，参数要求为字符串类型。

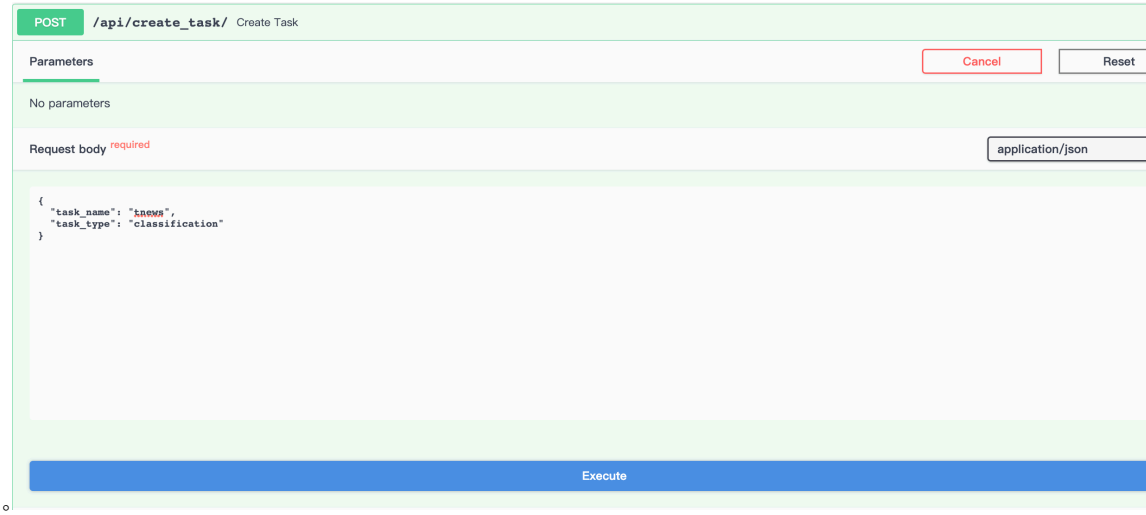
本示例中，task_name 设为 tnews, task_type 设为 classification，engine_type 设为 qiankunding

engine_type：只能设定为 qiankunding、bagualu 其中一种

task_type：若 engine_type 选择 qiankunding，则 task_type 只能选 classification、similarity、nli 一种；若 engine_type 选择 bagualu，则 task_type 只能选 ie、classification、summary 中一种

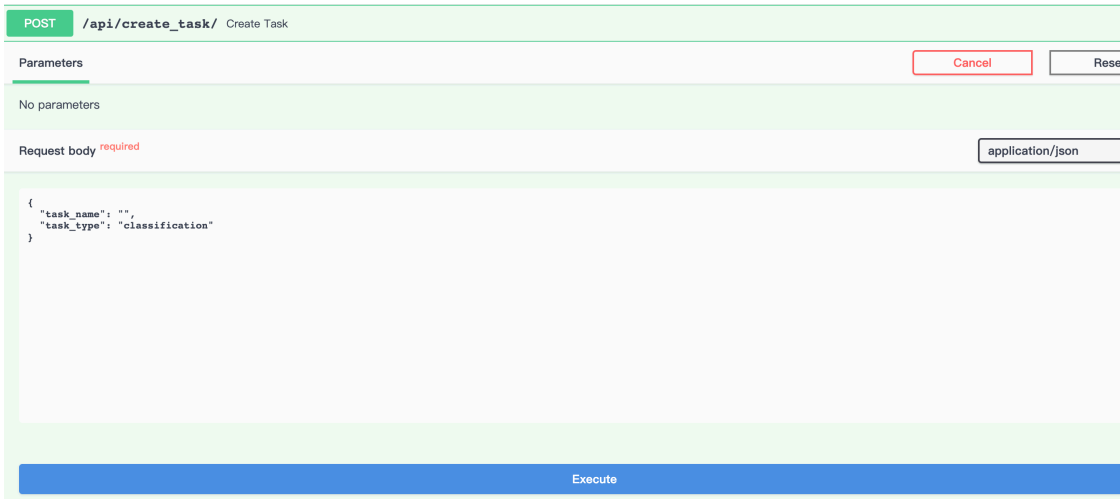
注：task_name 可以选填，为了方便后续任务管理，建议设置 task_name 参数，会生成与 task_name 同名的 task_id，如果不设置 task_name 参数，会自动生成 task_id，以下展示两种方式：

- 设置 task_name 参数。



- 点击 `Execute` 按钮执行，运行成功，会返回如下结果，其中 task_id 的值是 tnews，为我们指定的 task_name 的参数。

- 不设置 task_name 参数



点击 `Execute` 按钮执行，运行成功，会返回如下结果，其中 task_id 的值是由 task_type 与当前时间生成。

Curl

```
curl -X 'POST' \
  'http://localhost:5201/api/create_task/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task_name": "",
  "task_type": "classification"
}'
```

Request URL

```
http://localhost:5201/api/create_task/
```

Server response

Code	Details
200	Response body

```
{
  "ret_code": 200,
  "message": "task成功创建",
  "task_id": "classification_20221104114148"
}
```

Download

Response headers

```
content-length: 87
content-type: application/json
date: Fri,04 Nov 2022 03:41:47 GMT
server: uvicorn
```

Responses

avartar

2.2.3 查看任务列表

点击 Try it out 按钮，此接口不需设置参数，点击 Execute 按钮执行，将返回已运行完成和正在运行的所有任务，本示例中，可以看到有两个刚刚创建的任务。

Curl

```
curl -X 'POST' \
  'http://localhost:5201/api/list_task/' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

```
http://localhost:5201/api/list_task/
```

Server response

Code	Details
200	Response body

```
{
  "ret_code": 200,
  "message": "Success",
  "tasks": [
    "classification_20221104114148",
    "tnews"
  ]
}
```

Download

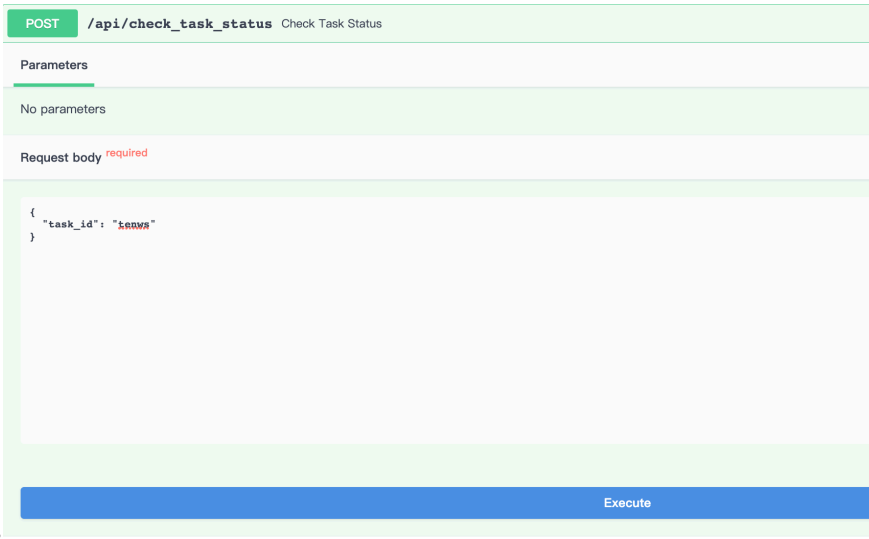
Response headers

```
content-length: 86
content-type: application/json
date: Fri,04 Nov 2022 03:49:05 GMT
server: uvicorn
```

Responses

avartar

2.2.4 查看任务状态



点击 `Try it out` 按钮,设置任务 id 参数为 tnews。

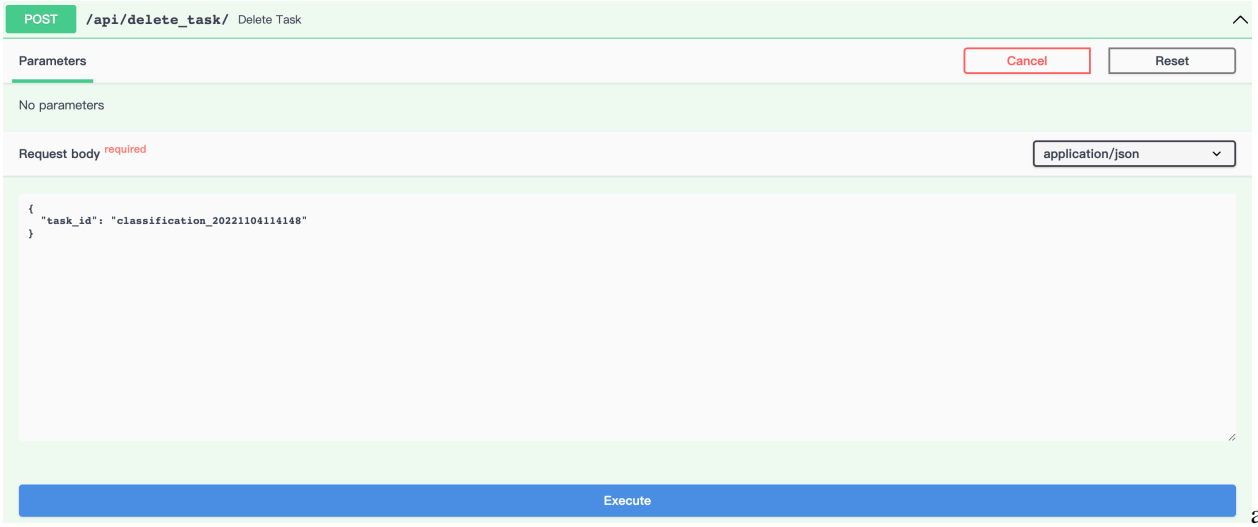
点击 `Execute` 执行，因为还未执行 tnews 任务，可以看到 tnews 任务处于”Initialized”的初始状态。

avartar

2.2.5 删除任务

点击 `Try it out`,设置任务 id 参数为为想要删除的任务 id，本示例删除任务 classification_20221104114148。

（注：会删除掉此任务下上传的数据以及训练好的模型的所有相关文件）



avartar

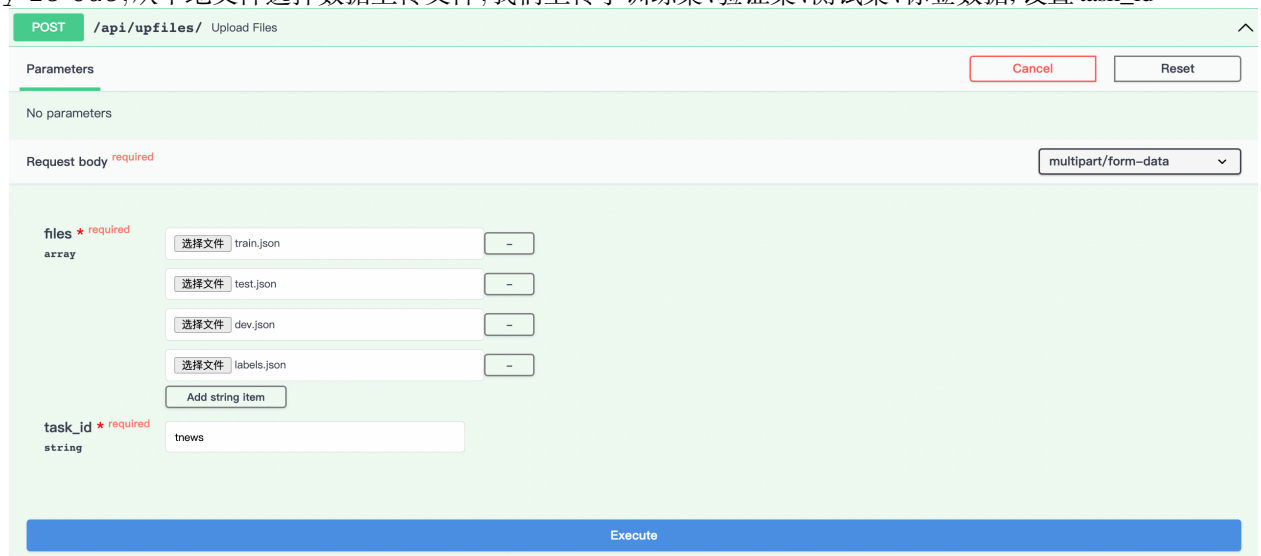
点击 `Execute` 执行,成功返回如下结果:

执行查看任务列表的接口,可以看到 classification_20221104114148 任务已不存在。

avartar

2.2.6 上传数据

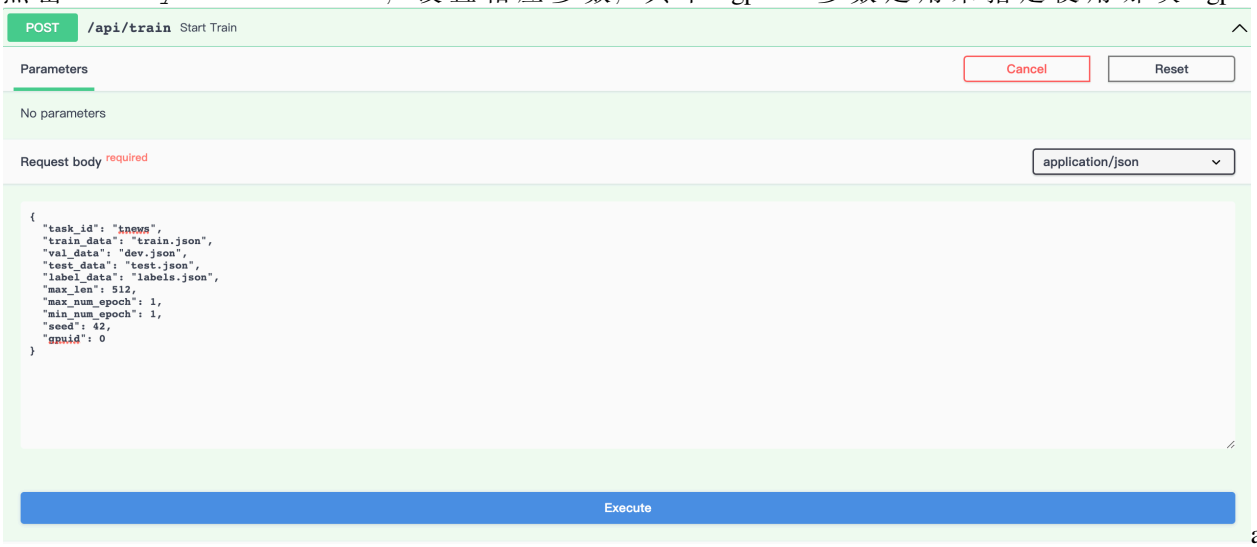
点击 `Try it out`，从本地文件选择数据上传文件，我们上传了训练集、验证集、测试集、标签数据，设置 task_id 为 tnews。avartar



点击 Execute 即上传数据：

2.2.7 启动训练

点击 Try it out，设置相应参数，其中 gpuid 参数是用来指定使用哪块 gpu：



avartar

点击 Execute,后台启动运行训练：

运行查看任务状态接口，可查看训练时的状态，下图显示模型正在训练中：

Curl

```
curl -X 'POST' \
  'http://localhost:5201/api/check_task_status' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task_id": "tnews"
}'
```

Request URL

```
http://localhost:5201/api/check_task_status
```

Server response

Code	Details
200	Response body

```
{
  "ret_code": 1,
  "message": "On Training"
}
```

Download

Response headers

```
content-length: 38
content-type: application/json
date: Fri,04 Nov 2022 04:03:48 GMT
server: uvicorn
```

Responses

avartar

下图显示模型已训练成功：

Curl

```
curl -X 'POST' \
  'http://localhost:5201/api/check_task_status' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task_id": "tnews"
}'
```

Request URL

```
http://localhost:5201/api/check_task_status
```

Server response

Code	Details
200	Response body

```
{
  "ret_code": 2,
  "message": "Train Success"
}
```

Response headers

```
content-length: 40
content-type: application/json
date: Fri,04 Nov 2022 04:07:40 GMT
server: uvicorn
```

Responses

2.2.8 停止训练

POST /api/stop_train Stop Train

Parameters

No parameters

Request body required

```
{
  "task_id": "tnews"
}
```

Execute

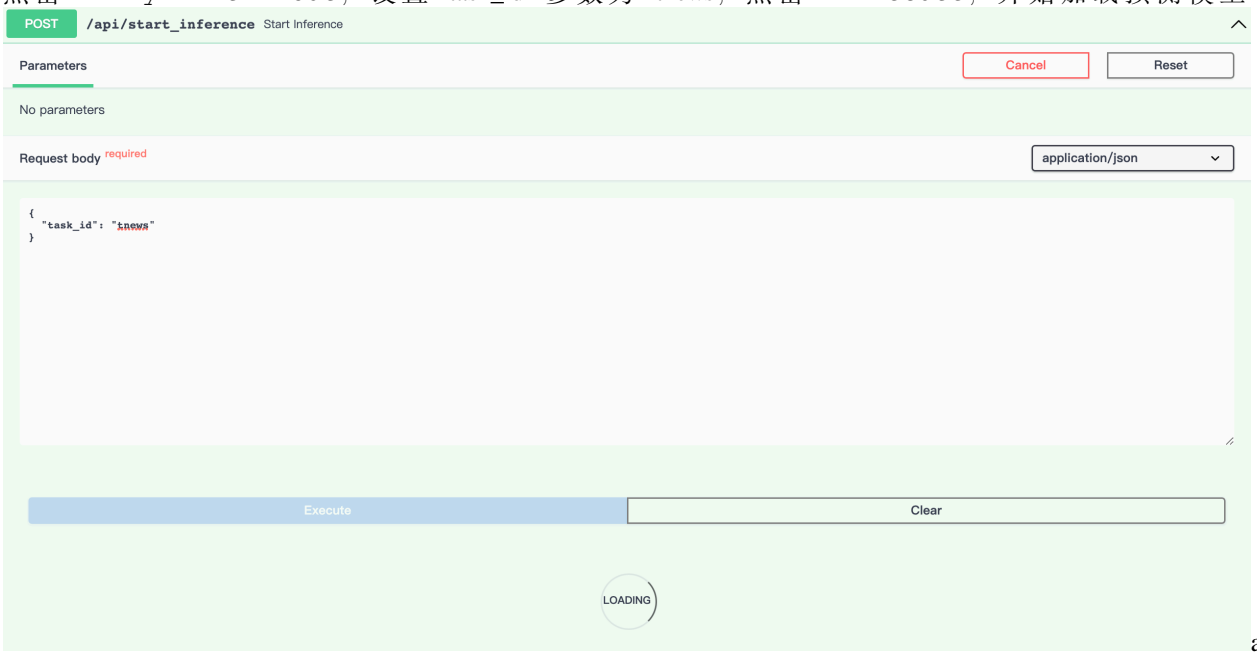
点击 `Try it out`，设置 task_id 参数为 tnews：

点击 `Execute`,tnews 的训练任务会被终止。

2.2.9 开启推理

模型训练成功，可以开启该任务的预测功能，预测样本要首先运行该接口。

点击 `Try it out`，设置 task_id 参数为 tnews，点击 `Execute`，开始加载预测模型:



avartar

模型加载成功：

Curl

```
curl -X 'POST' \
  'http://192.168.190.63:5201/api/start_inference' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task_id": "tnews"
}'
```

Request URL

```
http://192.168.190.63:5201/api/start_inference
```

Server response

Code	Details
200	Response body

```
{
  "ret_code": 200,
  "message": "加载预测模型"
}
```

Download

Response headers

```
content-length: 47
content-type: application/json
date: Thu,03 Nov 2022 13:54:12 GMT
server: uvicorn
```

Responses

av

运行查看任务状态接口，下图显示任务已开启预测模式：

Curl

```
curl -X 'POST' \
  'http://localhost:5201/api/check_task_status' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task_id": "tnews"
}'
```

Request URL

```
http://localhost:5201/api/check_task_status
```

Server response

Code	Details
200	Response body

```
{
  "ret_code": 2,
  "message": "On Inference"
}
```

Response headers

```
content-length: 39
content-type: application/json
date: Fri,04 Nov 2022 04:42:02 GMT
server: uvicorn
```

Responses

2.2.10 推理

点击 `Try it out`，设置对应参数，sentences 对应的是列表，列表中每个元素是字符串形式的待预测样本：

POST /api/predict Predict

Parameters

Cancel Reset

No parameters

Request body required

application/json

```
{
  "sentences": [
    "怎样的房子才算户型方正？",
    "文登区这些公路及危桥将进入封闭施工，请注意绕行！"
  ],
  "task_id": "tnews"
}
```

Execute

avartar

点击 `Execute`,会返还如下预测结果:

Request URL

http://192.168.190.63:5201/api/predict

Server response

Code Details

200

Response body


```
{
  "ret_code": 200,
  "result": {
    "predictions": [
      "房产",
      "汽车"
    ],
    "probabilities": [
      [
        0.001887392834760248 7,
        0.0032715012785047293,
        0.0009571036789566278,
        0.0006249119760468602,
        0.013577685691416264,
        0.9605872631072998,
        0.003316140966489911,
        0.0025132368318736553,
        0.0028051624540239573,
        0.001764467335306108,
        0.0009420909336768091,
        0.0017774890875443816,
        0.0013371729291975498,
        0.0035293134860694 41,
        0.0011090277694165707
```


Response headers

```
content-length: 745
content-type: application/json
date: Thu,03 Nov 2022 14:18:17 GMT
server: uvicorn
```

Responses

2.2.11 结束推理

点击 `Try it out`,设置task_id参数:

POST /api/end_inference End Inference

Parameters

No parameters

Request body required

```
{
  "task_id": "tnews"
}
```

Execute

点击 Execute：

Curl

```
curl -X 'POST' \
  'http://192.168.190.63:5201/api/end_inference' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task_id": "tnews"
}'
```

Request URL

```
http://192.168.190.63:5201/api/end_inference
```

Server response

Code	Details
200	Response body

```
{
  "ret_code": 200,
  "message": "释放预测模型"
}
```

Download

Response headers

```
content-length: 47
content-type: application/json
date: Thu,03 Nov 2022 14:19:37 GMT
server: uvicorn
```

Responses

运行查看任务状态接口，下图显示任务结束推理，返回训练完成后的状态：

Curl

```
curl -X 'POST' \
  'http://localhost:5201/api/check_task_status' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task_id": "tnews"
}'
```

Request URL

```
http://localhost:5201/api/check_task_status
```

Server response

Code	Details
200	Response body

```
{
  "ret_code": 2,
  "message": "Train Success"
}
```

Download

Response headers

```
content-length: 40
content-type: application/json
date: Fri,04 Nov 2022 04:51:12 GMT
server: uvicorn
```

Responses

avartar

CHAPTER

THREE

环境需求

GTS Engine 目前支持两种使用方式：

- 使用 Docker 加载镜像，启动服务；（推荐）
- 直接克隆源码，在您自己的开发环境中启动；

3.1 硬件环境要求

- CPU
  - 支持 x86_64（或称作 x64、Intel 64、AMD64）架构，不支持 arm64 架构。
- 内存
  - 乾坤鼎需要 24G 或以上。(此为训练 Erlangshen-MegatronBert-1.3B 模型所需内存空间，训练轻量模型则可以更小。预测部署时空间消耗 10G 左右。)
  - 八卦炉需要 8G 或以上
- 存储空间
  - 乾坤鼎需要 24G 或以上。(此为训练 Erlangshen-MegatronBert-1.3B 标准模型所需硬盘空间，训练轻量模型则可以更小。预测部署时空间消耗 10G 左右)
  - 八卦炉需要 8G 或以上
- GPU
  - 训练 Erlangshen-MegatronBert-1.3B 的 GPU 是 GeForce RTX ™ 3090 及以上，NVIDIA Ampere 架构。更好的 V100,A100 显卡可以大幅提升训练速度。
- 显存
  - 乾坤鼎需要 24G 或以上
  - 八卦炉需要 8G 或以上

3.2 不同环境下的测试效果

以 fewCLUE 榜单上的 csldcp 和 iflytek 任务为例进行测试 (test acc 项是 test_public.json 的准确率)：

乾坤鼎测试结果：

八卦炉文本分类测试结果

八卦炉信息抽取测试结果

> 备注：zh_weibo/MSRA/OntoNote4/Resume 为 NER 任务，其中 MSRA 在原始数据下进行测试；SanWen/FinRE 作为实体关系联合抽取任务进行测试，非单一关系分类任务

八卦炉摘要生成测试结果

> 备注：以下为在 lcsts 训练集中随机挑选 10000 条作为训练样本的结果，验证集和测试集均为原数据 test.jsonl

3.3 软件环境要求

建议您使用我们打包好的 Docker 镜像，如果您需要直接使用源码，请使用 `python>=3.7+` 并且需要满足下列的软件安装依赖：

```
fastapi==0.86.0
numpy==1.22.3
psutil==5.8.0
pydantic==1.10.2
pynvml==11.0.0
pytorch_lightning==1.7.6
scikit_learn==1.1.3
setuptools==58.0.4
starlette==0.20.4
torch==1.11.0+cu113
tqdm==4.62.3
transformers==4.18.0
uvicorn==0.19.0
python-multipart==0.0.5
sentence-transformers==2.2.2
LAC==2.1.2
textda==0.1.0.6
```

CHAPTER

FOUR

数据预处理

4.1 数据集类型

训练任务中，GTS Engine 要求您至少提供三个数据集：

- 训练集
- 验证集
- 测试集（可选）
- 无标签数据集（可选）
- 标签集（信息抽取、摘要生成无需标签集）

4.2 数据处理

所有的 NLP 任务的所有数据集，文件必须为 UTF-8 格式。每行表示一个样本，不同任务的样本数据格式如下：

4.2.1 文本分类任务

- **训练数据**

每行是一个样本，采用 json 格式，数据字段必须含有`"content"`和`"label"`字段，`"content"`对应的是输入文本，`"label"`字段对应该文本的标签。

```
# 示例
{
    "content": " 佛山市青少年武术比赛开幕，291 名武术达人同台竞技", "label": " 教育"
}
```

- **验证数据**

验证数据与训练数据格式一致。

```
# 示例
{
    "content": " 王者荣耀：官方悄悄的降价了 4 个强势英雄，看来米莱狄要来", "label": " 电竞"
}
```

- **测试数据**

测试数据与训练数据格式一致。

```
# 示例
{
    "content": " 上联：草根登上星光道，怎么对下联？", "label": " 文化"
}
```

- **无标签数据**

每行是一个样本，采用 json 格式，数据字段必须含有"content" 字段。

```
{"content": " 挥不去的是记忆，留不住的是年华，拎不起的是失落"}
```

- **标签数据**

数据为 json 格式，只有一行数据，必须含有” labels” 字段，对应的是标签的列表集合。

```
# 示例
{
    "labels": [" 故事", " 文化", " 娱乐", " 体育", " 财经", " 房产", " 汽车", " 教育", " 科技
↪", " 军事", " 旅游", " 国际", " 股票", " 农业", " 电竞"]
}
```

4.2.2 句子对任务

句子对任务包含语义匹配和自然语言推理任务，这两个任务的 label 比较固定，因此不需要标签文件，只需要训练集/验证集/测试集文件。

- **训练数据**

```
# 示例

# nli 任务，候选标签为 ["entailment", "contradiction", "neutral"]
{"id": 0, "sentence1": " 七五期间开始，国家又投资将武汉市区的部分土堤改建为钢筋泥凝土防水墙",
↪"sentence2": " 八五期间会把剩下的土堤都改建完", "label": "neutral"}

# 语义匹配任务，候选标签为 ["0","1"]，"1" 表示两句话语义是匹配的
{"id": 3, "sentence1": " 陈情令好不好看", "sentence2": " 陈情令好看不好看", "label": "1"}
```

- **验证数据**

格式和训练数据一样

- **测试数据**

```
# 示例
# nli 任务
{"id": 0, "sentence1": " 村里还有一个土地庙, 里面也装饰得十分气派", "sentence2": " 村里只有一个土地庙"}
# 语义匹配任务
{"id": 8, "sentence1": " 你会玩什么游戏", "sentence2": " 还有什么游戏"}
```

4.2.3 信息抽取任务

- **训练数据**

每行是一个样本，采用 json 格式，数据字段必须含有"task"、"text"、"entity_list"/"spo_list"、"choice" 字段，其中：

```
# 实体识别示例
{
    "task": "实体识别",
    "text": "我们是受到郑振铎先生、阿英先生著作的启示，从个人条件出发，瞄准现代出版史研究的空白，重点集藏解放区、国民党毁禁出版物。",
    "entity_list": [
        {
            "entity_text": "郑振铎",
            "entity_type": "人名",
            "entity_index": [5, 8]
        }, {
            "entity_text": "阿英",
            "entity_type": "人名",
            "entity_index": [11, 13]
        }, {
            "entity_text": "国民党",
            "entity_type": "组织机构",
            "entity_index": [50, 53]
        }
    ],
    "choice": ["组织机构", "人名", "地点"]
}
# 关系抽取示例
{
    "task": "关系抽取",
```

(continues on next page)

(continued from previous page)

```
    "text": "在导师阵容方面，英达有望联手《中国喜剧王》选拔新一代笑星",
    "spo_list": [
        {
            "predicate": "嘉宾",
            "subject": {
                "entity_text": "中国喜剧王",
                "entity_type": "电视综艺",
                "entity_index": [15, 20]
            },
            "object": {
                "entity_text": "英达",
                "entity_type": "人物",
                "entity_index": [8, 10]
            }
        }
    ],
    "choice": [
        ["电视综艺", "嘉宾", "人物"],
        ["影视作品", "主演", "人物"],
        ["影视作品", "编剧", "人物"],
        ["景点", "所在城市", "城市"],
        ["娱乐人物", "获奖", "奖项"],
        ["企业", "董事长", "人物"],
        ["图书作品", "作者", "人物"],
        ["机构", "成立日期", "时间"],
        ["国家", "首都", "城市"]
    ]
}
```

- **验证数据**

验证数据与训练数据格式一致。

- **测试数据**

测试数据与训练数据格式一致。

- **无标签数据**

无标签数据除无需提供"entity_list" 和"spo_list" 外，其他字段与训练数据格式一致。

4.2.4 摘要生成任务

- **训练数据**

每行是一个样本，采用 json 格式，数据字段必须含有"text" 和"summary" 字段，其中："text" 为输入文本，"summary" 为摘要生成目标。

```
# 示例

{ "text": " 央行网站消息：中国人民银行发布了《中国金融稳定报告 (2014)》，对 2013 年我国金融体系的稳定状况进行了全面评估。报告要求加强金融机构风险处置机制建设，建立存款保险制度，完善市场化的金融机构退出机制，建立维护金融稳定的长效机制。", "summary": " 央行：加快推进利率市场化建立存款保险制度"}
```

- **验证数据**

验证数据与训练数据格式一致。

- **测试数据**

测试数据与训练数据格式一致。

CHAPTER

FIVE

接口详情

GTS Engine 的所有接口都是 HTTP POST 请求，同时也提供了一个 python SDK 版本，本节列出了 GTS Engine 各个任务的详细接口：

5.1 文本分类任务

5.2 接口详情

5.2.1 创建任务

`create_task(self, task_name: str, task_type: str, engine_type: str)`

- 输入参数
- 输出参数

函数的返回值是一个字典，字典中包含如下字段：

5.2.2 列出任务列表

- 输入参数: 空

函数的返回值是一个字典，字典中包含如下字段：

5.2.3 查看任务状态

- 输入参数
- 输出参数

5.2.4 删除任务

- 输入参数

函数的返回值是一个字典，字典中包含如下字段：

5.2.5 上传文件

- 输入参数

5.2.6 开始训练

- 输入参数

函数的返回值是一个字典，字典中包含如下字段：

5.2.7 终止训练

- 输入参数

函数的返回值是一个字典，字典中包含如下字段：

5.2.8 开始推理

- 输入参数

函数的返回值是一个字典，字典中包含如下字段：

- 输入参数
- 输出参数

函数的返回值是一个字典，字典中包含如下字段：

- 输入参数

函数的返回值是一个字典，字典中包含如下字段：

CHAPTER

SIX

自定义开发

您可以在我们代码的基础上进行二次开发，自定义您的任务 pipeline 或者增加新的任务。

6.1 增加新的任务

6.1.1 定义输入输出

首先，你需要自己定义新任务的输入格式和输出格式，目前 Engine 仅支持每行一个 json 数据的格式，您的输入输出也要满足这个要求。

以文本分类任务为例，定义**输入字段**为：

- 数据文件

```
{
    "content": "a sentence",
    "label": "a label"
}
```

- 标签文件

```
{
    "labels": ["label1", "label2", "label3"]
}
```

定义输出字段为：

```
{
    "predictions": ["pred_label1", "pred_label2"],
    "probabilities": [[...], [...]]
}
```

6.1.2 定义你的 pipeline

在 gts_engine/pipelines 模块中，以 `{engine_type}_{task_type}.py` 的命名规则新建一个新任务的文件，其中，`engine_type` 需要为 `qiankunding` 或者 `bagualu`。

在 `{engine_type}_{task_type}.py` 中，需要导入 registry 并用 registry 装饰你的 pipeline 函数。

同时，需要实现以下三个函数：

```
from gts_common.registry import PIPELINE_REGISTRY


@PIPELINE_REGISTRY.register(suffix=__name__)
def train_pipeline(args):
    # implement your task pipeline here

@PIPELINE_REGISTRY.register(suffix=__name__)
def prepare_inference(save_path):
    # implement your inference preparation codes here
    # return a dict named 'inference_suite' including all models and data your need␣
↪to do inference with
    inference_suite = {}
    return inference_suite

@PIPELINE_REGISTRY.register(suffix=__name__)
def inference(samples, inference_suite):
    # implement your inference codes here
    # return a dict including inference results
    result = {}
    return result
```

由于使用了装饰器将你的训练和推理的函数注册到了 `PIPELINE_REGISTRY`，在启动训练和推理的时候，我们会根据 `engine_type` 和 `task_type` 动态调用对应任务的训练和推理函数。

这样你就只需要实现自定义的训练和推理逻辑，就可以进行自定义扩展，无需关心外部调用细节。

train_pipeline

在这个函数中定义你的训练流水线，输入参数是 `gts_engine_train.py` 中通过 argparse 解析出来的参数，可以在 `args` 中定义你高阶的训练参数；

注意，需要将你推理时用到的配置、数据和模型统一都放到 `args.save_path` 路径下；

注意，训练数据和测试数据的格式定义即你自己定义的格式；

prepare_inference

在这个函数中，加载推理时需要的模型和其他数据，以一个字典的格式返回，你需要自己定义 `inference_suite` 中的 key-value，如：

```
inference_suite = {
    "model": model,
    "args": args,
}
```

inference

在这个函数中，实现你的推理逻辑；

`samples` 是一个 list，list 中的每个 item 是一个 dict，dict 的数据结构即你自己定义的数据文件的数据结构；

`inference_suite` 是 `preprare_inference` 函数的返回值，包括推理时需要的模型和其他数据；

返回值 `result` 是一个 dict，dict 的字段需要和你定义的输出字段一致；

至此，就可以开始通过启动 `gts_engine_service` 以及调用 GTS Engine Client 测试你自定义的功能了。

CHAPTER

SEVEN

常见问题

本节列出了使用 GTS Engine 的常见问题，如果您有更多的使用问题，欢迎在 Github 上给我们提 Issue，我们会及时跟进解决。

7.1 常见问题 1

TODO

7.2 常见问题 2

TODO

7.3 常见问题 3

TODO

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search