

LLaMA-Factory:

安装:

```
git clone --depth 1 https://github.com/hiyouga/LLaMA-Factory.git
cd LLaMA-Factory
pip install -e ".[torch,metrics]"
#如果出现环境冲突, 请尝试使用 pip install --no-deps -e . 解决
```

校验指令:

```
llamafactory-cli version
```

环境:

- 1.QLora需要下载bitsandbytes
- 2.FlashAttention_2需要下载flash-attention
- 3.其他依赖:

名称	描述
torch	开源深度学习框架 PyTorch, 广泛用于机器学习和人工智能研究中。
torch-npu	PyTorch 的昇腾设备兼容包。
metrics	用于评估和监控机器学习模型性能。
deepspeed	提供了分布式训练所需的零冗余优化器。
bitsandbytes	用于大型语言模型量化。
hqq	用于大型语言模型量化。
eetq	用于大型语言模型量化。
gptq	用于加载 GPTQ 量化模型。
awq	用于加载 AWQ 量化模型。
aqlm	用于加载 AQLM 量化模型。
vllm	提供了高速并发的模型推理服务。
galore	提供了高效全参微调算法。
badam	提供了高效全参微调算法。
qwen	提供了加载 Qwen v1 模型所需的包。
modelscope	魔搭社区, 提供了预训练模型和数据集的下载途径。
swanlab	开源训练跟踪工具 SwanLab, 用于记录与可视化训练过程
dev	用于 LLaMA Factory 开发维护。

数据集格式：

1.指令监督微调 (Instruct Tuning)

目的：通过让模型学习详细的指令及对应的回答，优化其在特定指令下的表现。

格式要求：

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令", "第一轮回答"],
      ["第二轮指令", "第二轮回答"]
    ]
  }
]
```

模型输入构建方式：

- 最终输入为：instruction\ninput
- 若有 system 字段，则作为系统提示词前置。
- history 中的历史对话也会被用于训练，包含多轮交互信息。

示例：

```
[
  {
    "instruction": "今天的天气怎么样？",
    "input": "",
    "output": "今天的天气不错，是晴天。",
    "history": [
      ["今天会下雨吗？", "今天不会下雨，是个好天气。"],
      ["今天适合出去玩吗？", "非常适合，空气质量很好。"]
    ]
  }
]
```

dataset_info.json 配置示例：

```
"指令监督微调数据集名称": {
  "file_name": "data.json",
  "columns": {
    "prompt": "instruction",
    "query": "input",
    "response": "output",
    "system": "system",
    "history": "history"
  }
}
```

💡 Tip

如果模型本身具备推理能力（如 Qwen3）而数据集不包含思维链，LLaMA-Factory 会自动为数据添加空思维链。当 `enable_thinking` 为 `True` 时（慢思考，默认），空思维链会添加到模型回答中并且计算损失，否则会添加到用户指令中并且不计算损失（快思考）。请在训练和推理时保持 `enable_thinking` 参数一致。

如果您希望训练包含思维链的数据时使用慢思考，训练不包含思维链的数据时使用快思考，可以设置 `enable_thinking` 为 `None`。但该功能较为复杂，请谨慎使用。

2. 预训练数据集 (Pretraining Dataset)

目的：使模型学习语言结构和通用知识，提升泛化能力。

格式要求：仅提供原始文本内容。

```
[
  {"text": "document"},
  {"text": "another document"}
]
```

模型输入构建方式：

- 只使用 `text` 列中的内容进行训练。

dataset_info.json 配置示例：

```
"预训练数据集名称": {
  "file_name": "data.json",
  "columns": {
    "prompt": "text"
  }
}
```

3. 偏好数据集 (Preference Dataset)

应用场景：奖励模型训练、DPO 训练、ORPO 训练等。

目的：通过对比“更优”和“更差”的回答，引导模型输出更符合人类偏好的结果。

格式要求：

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "chosen": "优质回答（必填）",
    "rejected": "劣质回答（必填）"
  }
]
```

dataset_info.json 配置示例：

```
"偏好数据集名称": {
  "file_name": "data.json",
  "ranking": true,
  "columns": {
    "prompt": "instruction",
    "query": "input",
    "chosen": "chosen",
    "rejected": "rejected"
  }
}
```

4.KTO 数据集（Knowledge Transfer Optimization）

目的：与偏好数据集类似，但每轮问答只提供一个布尔标签（true/false）表示反馈。

📄 格式要求：

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "kto_tag": "人类反馈 [true/false]（必填）"
  }
]
```

dataset_info.json 配置示例：

```
"KTO数据集名称": {
  "file_name": "data.json",
  "columns": {
    "prompt": "instruction",
    "query": "input",
    "response": "output",
    "kto_tag": "kto_tag"
  }
}
```

总结对照表

类型	必填字段	选填字段	特殊配置
指令监督微调	instruction, output	input, system, history	prompt, query, response, systepythonm, history
预训练	text	—	prompt
偏好数据集	instruction, chosen, rejected	input	prompt, query, chosen, rejected, ranking: true
KTO 数据集	instruction, output, kto_tag	input	prompt, query, response, kto_tag

WebUI:

启动命令：

```
llamafactory-cli webui
```

页面：

语言zh

模型名称LLAMA3-8B

模型路径本地模型的文件夹或 Hugging Face 的模型标识符。
./Meta-Llama-3-8B

微调方法lora

适配器路径

刷新适配器

高级设置

TrainEvaluate & PredictChatExport

训练阶段目前采用的训练方式。
Supervised Fine-Tuning

数据路径数据文件夹的路径。
data

数据集alpaca_zh_demo

预览数据集

学习率AdamW 优化器的初始学习率。
5e-5

训练轮数需要执行的训练总轮数。
1

最大梯度范数用于梯度裁剪的范数。
1.0

最大样本数每个数据集的最大样本数。
100000

计算类型是否使用混合精度训练。
fp16

截断长度输入序列分词后的最大长度。
1024

批处理大小每个 GPU 处理的样本数量。
1

梯度累积梯度累积的步数。
16

验证集比例验证集占全部样本的百分比。
0

学习率调度器学习率调度器的名称。
cosine


监督微调：

可以使用以下命令使用 `examples/train_lora/llama3_lora_sft.yaml` 中的参数进行微调：

```
llamafactory-cli train examples/train_lora/llama3_lora_sft.yaml
```

也可以通过追加参数更新 yaml 文件中的参数：

```
llamafactory-cli train examples/train_lora/llama3_lora_sft.yaml \
  learning_rate=1e-5 \
  logging_steps=1
```

 **Tip**

LLaMA-Factory 默认使用所有可见的计算设备。根据需求可通过 `CUDA_VISIBLE_DEVICES` 或 `ASCEND_RT_VISIBLE_DEVICES` 指定计算设备。

`examples/train_lora/llama3_lora_sft.yaml` 提供了微调时的配置示例。该配置指定了模型参数、微调方法参数、数据集参数以及评估参数等。您需要根据自身需求自行配置。

```
### examples/train_lora/llama3_lora_sft.yaml
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct

stage: sft
do_train: true
finetuning_type: lora
lora_target: all

dataset: identity, alpaca_en_demo
template: llama3
```

```

cutoff_len: 1024
max_samples: 1000
overwrite_cache: true
preprocessing_num_workers: 16

output_dir: saves/llama3-8b/lora/sft
logging_steps: 10
save_steps: 500
plot_loss: true
overwrite_output_dir: true

per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-4
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
ddp_timeout: 180000000

val_size: 0.1
per_device_eval_batch_size: 1
eval_strategy: steps
eval_steps: 500

```

Tip

模型 `model_name_or_path` 、数据集 `dataset` 需要存在且与 `template` 相对应。

名称	描述
model_name_or_path	模型名称或路径
stage	训练阶段， 可选: rm(reward modeling), pt(pretrain), sft(Supervised Fine-Tuning), PPO, DPO, KTO, ORPO
do_train	true用于训练, false用于评估
finetuning_type	微调方式。可选: freeze, lora, full
lora_target	采取LoRA方法的目标模块，默认值为 <code>all</code> 。
dataset	使用的数据集，使用“,”分隔多个数据集
template	数据集模板，请保证数据集模板与模型相对应。
output_dir	输出路径
logging_steps	日志输出步数间隔
save_steps	模型断点保存间隔
overwrite_output_dir	是否允许覆盖输出目录
per_device_train_batch_size	每个设备上训练的批次大小
gradient_accumulation_steps	梯度积累步数

名称	描述
max_grad_norm	梯度裁剪阈值
learning_rate	学习率
lr_scheduler_type	学习率曲线，可选 <code>linear</code> ， <code>cosine</code> ， <code>polynomial</code> ， <code>constant</code> 等。
num_train_epochs	训练周期数
bf16	是否使用 bf16 格式
warmup_ratio	学习率预热比例
warmup_steps	学习率预热步数
push_to_hub	是否推送模型到 Huggingface

Merge:

合并

当我们基于预训练模型训练好 LoRA 适配器后，我们不希望在每次推理的时候分别加载预训练模型和 LoRA 适配器，因此我们需要将预训练模型和 LoRA 适配器合并导出成一个模型，并根据需要选择是否量化。根据是否量化以及量化算法的不同，导出的配置文件也有所区别。

您可以通过 `llamafactory-cli export merge_config.yaml` 指令来合并模型。其中 `merge_config.yaml` 需要您根据不同情况进行配置。

`examples/merge_lora/llama3_lora_sft.yaml` 提供了合并时的配置示例。

```
### examples/merge_lora/llama3_lora_sft.yaml
### model
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct
adapter_name_or_path: saves/llama3-8b/lora/sft
template: llama3
finetuning_type: lora

### export
export_dir: models/llama3_lora_sft
export_size: 2
export_device: cpu
export_legacy_format: false
```

Tip

- 模型 `model_name_or_path` 需要存在且与 `template` 相对应。 `adapter_name_or_path` 需要与微调中的适配器输出路径 `output_dir` 相对应。
- 合并 LoRA 适配器时，不要使用量化模型或指定量化位数。您可以使用本地或下载的未量化的预训练模型进行合并。

量化

在完成模型合并并获得完整模型后，为了优化部署效果，人们通常会基于显存占用、使用成本和推理速度等因素，选择通过量化技术对模型进行压缩，从而实现更高效的部署。

量化（Quantization）通过数据精度压缩有效地减少了显存使用并加速推理。LLaMA-Factory 支持多种量化方法，包括：

- AQLM
- AWQ
- GPTQ
- QLoRA
- ...

GPTQ 等后训练量化方法(Post Training Quantization)是一种在训练后对预训练模型进行量化的方法。我们通过量化技术将高精度表示的预训练模型转换为低精度的模型，从而在避免过多损失模型性能的情况下减少显存占用并加速推理，我们希望低精度数据类型在有限的表示范围内尽可能地接近高精度数据类型的表示，因此我们需要指定量化位数 `export_quantization_bit` 以及校准数据集 `export_quantization_dataset`。

在进行模型合并时，请指定：

- `model_name_or_path`: 预训练模型的名称或路径
- `template`: 模型模板
- `export_dir`: 导出路径
- `export_quantization_bit`: 量化位数
- `export_quantization_dataset`: 量化校准数据集
- `export_size`: 模型导出时每个文件的最大大小（GB），超出该大小则自动分片保存。
- `export_device`: 导出设备
- `export_legacy_format`: 是否使用旧格式导出

下面提供一个配置文件示例：

```
### examples/merge_lora/llama3_gptq.yaml
### model
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct
template: llama3

### export
export_dir: models/llama3_gptq
export_quantization_bit: 4
export_quantization_dataset: data/c4_demo.json
export_size: 2
export_device: cpu
export_legacy_format: false
```

QLoRA 是一种在 4-bit 量化模型基础上使用 LoRA 方法进行训练的技术。它在极大地保持了模型性能的同时大幅减少了显存占用和推理时间。

⚠ Warning

不要使用量化模型或设置量化位数 `quantization_bit`

下面提供一个配置文件示例：

```
### examples/merge_lora/llama3_q_lora.yaml
### model
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct
adapter_name_or_path: saves/llama3-8b/lora/sft
template: llama3
finetuning_type: lora

### export
export_dir: models/llama3_lora_sft
export_size: 2
export_device: cpu
export_legacy_format: false
```

Inference

LLaMA-Factory 支持多种推理方式。

您可以使用 `llamafactory-cli chat inference_config.yaml` 或 `llamafactory-cli webchat inference_config.yaml` 进行推理与模型对话。对话时配置文件只需指定原始模型 `model_name_or_path` 和 `template`，并根据是否是微调模型指定 `adapter_name_or_path` 和 `finetuning_type`。

如果您希望向模型输入大量数据集并保存推理结果，您可以启动 `vllm` 推理引擎对大量数据集进行快速的批量推理。您也可以通过 部署 api 服务的形式通过 api 调用来进行批量推理。默认情况下，模型推理将使用 Huggingface 引擎。您也可以指定 `infer_backend: vllm` 以使用 `vllm` 推理引擎以获得更快的推理速度。

⚠ Warning

使用任何方式推理时，模型 `model_name_or_path` 需要存在且与 `template` 相对应。

原始模型推理配置

对于原始模型推理，`inference_config.yaml` 中只需指定原始模型 `model_name_or_path` 和 `template` 即可。

```
### examples/inference/llama3.yaml
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct
template: llama3
infer_backend: huggingface #choices: [huggingface, vllm]
```

微调模型推理配置

对于微调模型推理，除原始模型和模板外，还需要指定适配器路径 `adapter_name_or_path` 和微调类型 `finetuning_type`。

```
### examples/inference/llama3_lora_sft.yaml
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct
adapter_name_or_path: saves/llama3-8b/lora/sft
template: llama3
finetuning_type: lora
infer_backend: huggingface #choices: [huggingface, vllm]
```

多模态模型

对于多模态模型，您可以运行以下指令进行推理。

```
llamafactory-cli webchat examples/inference/llava1.5.yaml
```

`examples/inference/llava1.5.yaml` 的配置示例如下：

```
model_name_or_path: llava-hf/llava-1.5-7b-hf
template: vicuna
infer_backend: huggingface #choices: [huggingface, vllm]
```

批量推理

数据集

您可以通过以下指令启动 vllm 推理引擎并使用数据集进行批量推理：

```
python scripts/vllm_infer.py --model_name_or_path path_to_merged_model --dataset alpaca_en_demo
```

api

如果您需要使用 api 进行批量推理，您只需指定模型、适配器（可选）、模板、微调方式等信息。

下面是一个配置文件的示例：

```
### examples/inference/llama3_lora_sft.yaml
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct
adapter_name_or_path: saves/llama3-8b/lora/sft
template: llama3
finetuning_type: lora
```

下面是一个启动并调用 api 服务的示例：

您可以使用 `API_PORT=8000 CUDA_VISIBLE_DEVICES=0 llamafactory-cli api examples/inference/llama3_lora_sft.yaml` 启动 api 服务并运行以下示例程序进行调用：

```
# api_call_example.py
from openai import OpenAI
client = OpenAI(api_key="0",base_url="http://0.0.0.0:8000/v1")
messages = [{"role": "user", "content": "who are you?"}]
result = client.chat.completions.create(messages=messages, model="meta-llama/Meta-Llama-3-8B-Instruct")
print(result.choices[0].message)
```

Eval:

通用能力评估

在完成模型训练后，您可以通过 `llamafactory-cli eval`
`examples/train_lora/llama3_lora_eval.yaml` 来评估模型效果。

配置示例文件 `examples/train_lora/llama3_lora_eval.yaml` 具体如下：

```
### examples/train_lora/llama3_lora_eval.yaml
### model
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct
adapter_name_or_path: saves/llama3-8b/lora/sft # 可选项

### method
finetuning_type: lora

### dataset
task: mmlu_test # mmlu_test, ceval_validation, cmmlu_test
template: fewshot
lang: en
n_shot: 5

### output
save_dir: saves/llama3-8b/lora/eval

### eval
batch_size: 4
```

NLG 评估

此外，您还可以通过 `llamafactory-cli train`
`examples/extras/nlg_eval/llama3_lora_predict.yaml` 来获得模型的 BLEU 和 ROUGE 分数以评价模型生成质量。

配置示例文件 `examples/extras/nlg_eval/llama3_lora_predict.yaml` 具体如下：

```
### examples/extras/nlg_eval/llama3_lora_predict.yaml
### model
model_name_or_path: meta-llama/Meta-Llama-3-8B-Instruct
adapter_name_or_path: saves/llama3-8b/lora/sft

### method
stage: sft
do_predict: true
finetuning_type: lora

### dataset
eval_dataset: identity, alpaca_en_demo
template: llama3
cutoff_len: 2048
max_samples: 50
overwrite_cache: true
preprocessing_num_workers: 16
```

```
### output
output_dir: saves/llama3-8b/lora/predict
overwrite_output_dir: true

### eval
per_device_eval_batch_size: 1
predict_with_generate: true
ddp_timeout: 180000000
```

同样，您也通过在指令 `python scripts/vllm_infer.py --model_name_or_path path_to_merged_model --dataset alpaca_en_demo` 中指定模型、数据集以使用 vllm 推理框架以取得更快的推理速度。

评估相关参数

参数名称	类型	介绍
task	str	评估任务的名称，可选项有 mmlu_test, ceval_validation, cmmlu_test
task_dir	str	包含评估数据集的文件夹路径，默认值为 <code>evaluation</code> 。
batch_size	int	每个GPU使用的批量大小，默认值为 <code>4</code> 。
seed	int	用于数据加载器的随机种子，默认值为 <code>42</code> 。
lang	str	评估使用的语言，可选值为 <code>en</code> 、 <code>zh</code> 。默认值为 <code>en</code> 。
n_shot	int	few-shot 的示例数量，默认值为 <code>5</code> 。
save_dir	str	保存评估结果的路径，默认值为 <code>None</code> 。如果该路径已经存在则会抛出错误。
download_mode	str	评估数据集的下载模式，默认值为 <code>DownloadMode.REUSE_DATASET_IF_EXISTS</code> 。如果数据集已经存在则重复使用，否则则下载。