

主题

llama介绍、运行、部署、推理、微调

介绍LLaMA

Q: 什么是LLaMA?

A:

基本介绍

- 2023年2月，Meta推出LLaMA（Large Language Model Meta AI），训练并发布了一系列模型，模型参数有7B（70亿）、13B（130亿）、30B（300亿）、65B（650亿）。
- 2023年7月，Meta推出LLaMA2，训练并发布了三种模型大小：7B、13B和70B。
 - 模型可商用。
 - 模型架构与Llama 1模型基本保持不变，但用于训练基础模型的数据增加了40%。
 - 随附的预印本还提到了一个具有34B参数的模型，该模型可能在未来满足安全目标后发布。
 - Llama 2包括基础模型和针对对话进行微调的模型，称为Llama 2 - chat。

from [LLaMA维基百科](#)

相关资料

llama v1 相关资料：

官方介绍：<https://research.facebook.com/publications/llama-open-and-efficient-foundation-language-models/>

论文：[《LLaMA: Open and Efficient Foundation Language Models》](#)

github仓库：https://github.com/meta-llama/llama/tree/llama_v1 (llama_v1分支)

llama v2 相关资料：

官方介绍：<https://about.fb.com/news/2023/07/llama-2/>

论文：[《Llama 2: Open Foundation and Fine-Tuned Chat Models》](#)

github仓库：<https://github.com/meta-llama/llama> (当前默认llama v2版本)

论文分析

llama v1

论文-**Abstract**里提到：

- LLaMA，是一组基础语言模型，参数范围从7B到65B。
- 模型在数万亿个token上训练
- 可以使用公开可用的数据集来训练模型
- 特别是，LLaMA-13B在大多数基准测试中的表现都优于GPT-3(175B)，而LLaMA-65B与最好的模型Chinchilla70B和PaLM-540B相比也具有竞争力。

- 向研究界发布所有模型
 - 开源模型代码
 - 开源模型权重（需要发邮件申请）

论文-1 **Introduction**里提到：

The focus of this work is to train a series of language models that achieve the best possible performance at various inference budgets, by training on more tokens than what is typically used. The resulting models, called LLaMA, ranges from 7B to 65B parameters with competitive performance compared to the best existing LLMs.

这项工作的重点是通过训练更多的token，训练一系列语言模型，在各种推理预算下实现尽可能最佳的性能。由此产生的模型，称为LLAMA，参数范围从7B到65B，与现有最佳的LLM相比，具有竞争力的性能。

所以，LLAMA的提出是考虑的是这样一种问题：推理预算有限的情况下，怎样达到较好的推理结果？

我们都知道超大参数量的大模型的效果很好，但我们没有超强的算力去训练、去推理，那是属于大公司的工作。那llama的提出就是告诉开发者，即使模型规模小，资源有限，如果选择合适的训练方法，用足够的多高质量的数据训练足够长的时间，依旧可以得到一个能力很好的模型。

论文-1 **Introduction**里还提到一些细节：

- 预训练的数据全部来自公开可用的数据集（publicly available data），不像有的大模型使用了一些不公开的数据
- LLaMA-13B在大多数基准测试上优于GPT-3（175B），并且可以在单GPU可运行

论文-2.1 **Pre-training Data**里提到：

训练数据的组成：

<https://github.com/Eric-LLMs>

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

该模型主要针对英语进行了训练，但也针对其他一些使用拉丁或西里尔字母的语言进行了训练。

例如，在维基百科上对以下 20 种语言进行了 LLaMA 训练：bg、ca、cs、da、de、en、es、fr、hr、hu、it、nl、pl、pt、ro、ru、sl、sr、sv、uk。

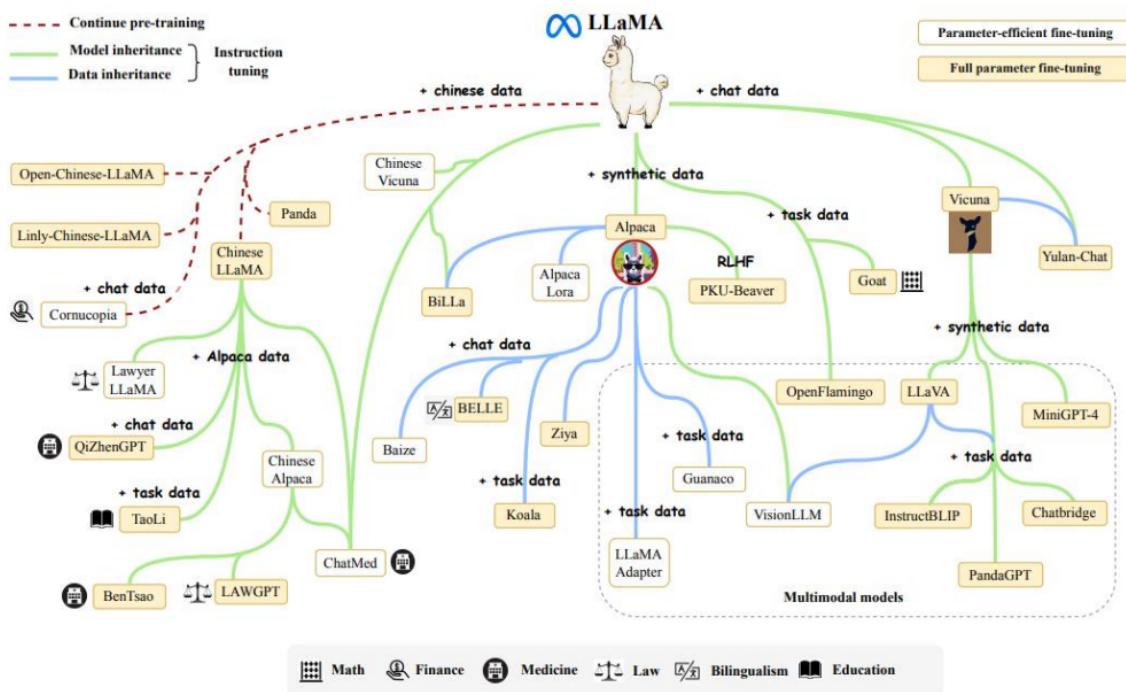
LaMA 的分词器会将未识别的字符分割成 UTF-8 字节，因此，它也可以处理其他语言，如中文或日文，尽管它们使用不同的字符。

虽然这些语言在培训中的比例可以忽略不计，但LLaMA仍然展示了一些中英翻译的能力：

```
Prompt = "J'aime le chocolat = I like chocolate\n祝你一天过得愉快 ="  
Output = "I wish you a nice day"
```

LLaMA模型家族

LLaMA model family



简单解读LLaMA源码

llama v1

<https://github.com/meta-llama/llama>

requirements.txt

torch
fairscale
fire
sentencepiece

1. **torch**: 这是 PyTorch 深度学习框架的标准包。PyTorch 是一个用于构建动态神经网络的开源机器学习库，它提供了强大的张量计算功能以及方便的神经网络构建接口。
2. **fairscale**: 这是 Facebook AI Research (FAIR) 开发的 PyTorch 生态系统的一个扩展库。它提供了一些工具和模块，用于支持大规模、高效的分布式训练和模型并行化。
3. **fire**: 这是一个 Python 库，用于生成命令行界面 (CLI)。它可以根据 Python 对象自动生成 CLI，使得编写命令行工具变得更加简单和直观。
4. **sentencepiece**: 这是一个用于生成和处理文本片段的 Python 库。它提供了一种分割和编码文本的方法，尤其在自然语言处理任务中常用于分词、标记化和文本编码。

setup.py

这个 setup.py 文件也是一个 Python 包的配置文件，用于定义该包的元数据和包含的子包。让我们来看看它的内容：

版权声明: 文件开头包含了版权声明，说明了该软件的版权归原作者所有以及使用和分发的许可条款。根据注释，该软件受到 GNU General Public License version 3 (GNU GPL-3.0) 的许可协议的约束。GNU GPL-3.0 是一种自由软件许可证，要求任何修改或衍生的软件必须以相同的许可证发布。

导入模块: 使用 from setuptools import setup, find_packages 导入了 setuptools 包中的两个函数: setup 和 find_packages。find_packages 函数用于查找包中的所有子包，setup 函数用于配置和注册该包。

setup 函数: 使用 setup 函数来配置该 Python 包的元数据。重要的参数包括：

name: 包的名称，在这里是 "llama"。

version: 包的版本号，在这里是 "0.0.0"。

packages: 通过 find_packages() 函数来查找所有包，并包含在打包过程中。这样可以确保所有子包都被正确包含。

通过这个 setup.py 文件，可以方便地使用 setuptools 来构建、安装和分发该 Python 包，并且根据 GNU GPL-3.0 许可证规定进行分发。

example.py

运行模型的命令

```
torchrun --nproc_per_node MP example.py --ckpt_dir $TARGET_FOLDER/model_size --  
tokenizer_path $TARGET_FOLDER/tokenizer.model
```

这段文本提供了关于如何在单个或多个 GPU 节点上运行提供的 example.py 脚本以执行推理任务的说明。

首先，通过使用 torchrun 命令，可以在 PyTorch 中运行分布式训练脚本。torchrun 命令的一些参数和选项如下：

- `--nproc_per_node MP`: 指定每个节点上使用的 GPU 数量。在这里，`MP` 可以是一个整数，表示要使用的 GPU 数量。如果 `MP` 被设置为 `auto`，系统将尝试自动确定 GPU 数量。这样可以在单个或多个 GPU 节点上进行并行计算。

然后，`example.py` 是要执行的 Python 脚本的文件名。在这个例子中，它是一个推理脚本，用于生成文本的完成。

接下来是两个参数：

- `--ckpt_dir $TARGET_FOLDER/model_size`: 指定了模型的检查点目录，其中 `$TARGET_FOLDER` 是一个环境变量，它指向了模型的文件夹。检查点目录应该包含了预训练模型的权重和其他必要文件。
- `--tokenizer_path $TARGET_FOLDER/tokenizer.model`: 指定了分词器模型的路径，其中 `$TARGET_FOLDER` 同样是一个环境变量，指向了分词器模型的文件。分词器用于将文本转换为模型可以理解的输入表示。

综合来看，这段文本提供了一个示例命令，用于在单个或多个 GPU 节点上运行 `example.py` 脚本，并指定了模型的检查点目录和分词器模型的路径。

源码解释

这个 Python 脚本实现了加载预训练的 LLaMA 模型，并用于生成文本的功能。让我们逐步解释脚本的各个部分：

1. **版权声明**: 脚本开头包含了版权声明，指明了脚本的版权信息和许可条款。
2. **导入模块**: 脚本导入了一些 Python 模块，这些模块包括了用于文件操作、系统交互、深度学习模型和模型并行化的库。例如，`torch` 是一个用于深度学习的库，`fire` 是一个用于创建命令行接口的库。
3. **设置模型并行**: 脚本定义了一个函数用于设置模型并行环境。这个函数会获取本地进程的排名和世界大小，并初始化模型并行环境。这个函数在多 GPU 环境中用于协调模型的并行运行。
4. **加载模型**: 脚本定义了一个函数用于加载预训练的 LLaMA 模型。这个函数会读取模型的参数文件和权重文件，并初始化生成器。加载模型的过程包括了读取参数、初始化 Tokenizer 和 Transformer 模型，并加载模型的权重。
5. **主函数 main**: 主函数是整个脚本的入口点。它调用了设置模型并行的函数和加载模型的函数，并提供了命令行接口用于用户指定模型路径、生成参数等。主函数会生成一系列文本，并打印输出结果。

总的来说，这个脚本实现了一个通过预训练的 LLaMA 模型生成文本的功能，用户可以通过命令行接口指定参数并生成相应的文本。

翻译prompts

1. "I believe the meaning of life is"
 - 我相信生命的意义是
2. "Simply put, the theory of relativity states that"
 - 简而言之，相对论理论指出
3. "Building a website can be done in 10 simple steps:\n"
 - 建立一个网站可以分为 10 个简单的步骤：
4. "Tweet: "I hate it when my phone battery dies."

Sentiment: Negative

###

Tweet: "My day has been  "

Sentiment: Positive

###

Tweet: "This is the link to the article"

Sentiment: Neutral

###

Tweet: "This new music video was incredibile"

Sentiment:

- 推文：“我讨厌我的手机电池没电时的感觉。”

情感：负面

推文：“我的一天过得很棒”

情感：积极

推文：“这是文章的链接”

情感：中性

推文：“这个新的音乐视频太令人难以置信了”

情感：

5. "Translate English to French:

sea otter => loutre de mer

peppermint => menthe poivrée

plush girafe => girafe peluche cheese =>"

- 将英语翻译成法语：

海獭 => 海獭

薄荷 => 薄荷

绒毛长颈鹿 => 绒毛长颈鹿

奶酪 =>

Q：这些例子考察了模型的什么能力？

这些例子涵盖了多个不同方面的文本生成任务，以评估模型在各种情况下的能力。具体来说：

1. **生命意义的探讨：**这个提示要求模型提供对生命意义的看法，这是一个开放性的哲学问题，需要模型具有理解人类价值观和情感的能力，以及能够进行逻辑推理和文本生成的能力。
2. **相对论的解释：**这个提示要求模型简要解释相对论的理论，这需要模型具备对科学概念的理解，以及能够用自然语言清晰表达复杂概念的能力。
3. **建立网站的步骤：**这个提示提供了一个任务描述，要求模型列出建立网站的步骤。这需要模型具备技术性知识的理解，以及能够提供清晰步骤的表达能力。
4. **推特情感分析：**这个提示提供了一些推特内容，并要求模型对推文的情感进行分析。这需要模型具备对语言情感的理解和推断能力。
5. **英法翻译：**这个提示要求模型将一些英文单词翻译成法语。这需要模型具备语言翻译的能力，包括对单词意义的理解和正确的翻译能力。

综上所述，这些提示涵盖了模型在理解、推理、生成各种类型文本、语言理解和翻译等方面的能力。通过这些例子，可以评估模型在不同任务和情境下的表现。

[download.sh](#)

这个文件是一个 Bash 脚本，用于从指定的预签名 URL 下载模型文件。下面是对这个文件的详细解释：

1. **版权声明：**文件开头声明了版权信息，指出了该软件可以根据 GNU General Public License version 3 进行使用和分发。
2. **变量定义：**
 - `PRESIGNED_URL`：从电子邮件中获得的预签名 URL，用于下载模型文件。
 - `MODEL_SIZE`：需要下载的模型大小列表，用逗号分隔。可以根据需要编辑这个列表，指定要下载的模型大小。
 - `TARGET_FOLDER`：文件将下载到的目标文件夹路径。
3. **N_SHARD_DICT 字典：**

- 这是一个关联数组，用于存储每种模型大小对应的分片数。每种模型大小对应的分片数用于循环下载相应数量的模型分片文件。

4. 下载分词器文件:

- 使用 `wget` 命令从预签名 URL 下载分词器文件 (`tokenizer.model`) 和检查文件 (`tokenizer_checklist.chk`) 到目标文件夹。
- 使用 `md5sum -c` 命令验证下载的文件的 MD5 校验和。

5. 循环下载模型文件:

- 对于 `MODEL_SIZE` 中的每个模型大小，执行以下步骤：
 - 创建相应模型大小的文件夹。
 - 根据 `N_SHARD_DICT` 中的分片数，使用循环下载模型分片文件。
 - 下载参数文件 (`params.json`) 和检查文件 (`checklist.chk`)。
 - 使用 `md5sum -c` 命令验证下载的文件的 MD5 校验和。

通过这些步骤，该脚本可以从预签名 URL 下载指定的模型文件，并验证其完整性。

模型相关文件组织形式

如果成功下载了所有文件，当前文件夹中的文件组织形式将如下所示：

```
- tokenizer.model
- tokenizer_checklist.chk
- 7B/
  - consolidated.0.pth
  - params.json
  - checklist.chk
- 13B/
  - consolidated.0.pth
  - consolidated.1.pth
  - params.json
  - checklist.chk
- 30B/
  - consolidated.0.pth
  - consolidated.1.pth
  - consolidated.2.pth
  - consolidated.3.pth
  - params.json
  - checklist.chk
- 65B/
  - consolidated.0.pth
  - consolidated.1.pth
  - consolidated.2.pth
  - consolidated.3.pth
  - consolidated.4.pth
  - consolidated.5.pth
  - consolidated.6.pth
  - consolidated.7.pth
  - params.json
  - checklist.chk
```

llama/generation.py

定义了Llama类

这段代码是一个名为 LLaMA 的 Python 类，定义了一个文本生成器。让我们来逐步解读这段代码：

1. **版权声明:** 和之前的文件一样，这里也有版权声明，说明了软件的版权归属和使用许可条款。
2. **导入模块:** 这个文件导入了 `typing` 模块中的 `List` 类型，以及 `torch` 库。`typing` 模块中的 `List` 类型用于声明列表类型，`torch` 库是 PyTorch 深度学习库。
3. **LLaMA 类:** 这个类包含了以下方法和属性：
 - **`__init__` 方法:** 初始化方法，接受两个参数 `model` 和 `tokenizer`，分别代表了 Transformer 模型和分词器对象。在初始化过程中将这两个参数保存为对象的属性。
 - **`generate` 方法:** 用于生成文本的主要方法，接受一系列的提示 (`prompts`)、最大生成长度 (`max_gen_len`)、温度 (`temperature`) 和 top-p 参数 (`top_p`)。该方法的主要逻辑是利用给定的模型和分词器对象来生成文本。它首先对提示进行编码，然后逐步生成文本，并根据温度和 top-p 参数来调整生成的随机性。最终返回生成的文本列表。
 - **`sample_top_p` 函数:** 用于根据给定的概率分布 `probs` 和 top-p 参数 `p` 来采样下一个 token。它首先对概率分布进行排序，并计算累积概率，然后根据 top-p 参数来筛选出下一个 token。最后返回采样的下一个 token。

这个类实现了一个简单的文本生成器，它可以根据给定的提示生成文本，并可以控制生成过程的随机性。

llama/model.py

定义了ModelArgs类、RMSNorm类、Attention类、FeedForward类、TransformerBlock类、Transformer类

LLaMA模型是基于Transformer架构的基础上进行改进的，所以这份代码主要描绘了一些模型的构建细节。

论文 - 2.2 Architecture 里提到的几个改进，（后面的括号[]代表从哪里找到的灵感）：

- Pre-normalization [GPT3]
- SwiGLU activation function [PaLM]
- Rotary Embeddings [GPTNeo]

这段代码定义了一个 Transformer 模型，用于生成文本。让我们逐步解读：

1. **版权声明:** 和之前的文件一样，这里也有版权声明，说明了软件的版权归属和使用许可条款。
2. **导入模块:** 这个文件导入了一些 Python 标准库中的模块，比如 `typing` 和 `dataclasses`，以及一些 PyTorch 相关的模块。
3. **ModelArgs 数据类:** 这个数据类用于存储模型的参数，包括维度 (`dim`)、层数 (`n_layers`)、注意力头数 (`n_heads`)、词汇表大小 (`vocab_size`) 等。它还包含了一些默认值，比如 `max_batch_size` 和 `max_seq_len`。
4. **RMSNorm 类:** 这是一个自定义的 PyTorch 模块，用于实现 RMS 归一化。
5. **`precompute_freqs_cis` 函数:** 这个函数用于预算旋转的频率。它会根据给定的维度和长度来计算旋转的频率。

6. `reshape_for_broadcast` 函数: 这个函数用于重新调整频率张量的形状，以便进行广播。
7. `apply_rotary_emb` 函数: 这个函数用于应用旋转嵌入。
8. `Attention` 类: 这个类定义了注意力机制的实现。它包含了注意力的前向传播逻辑，以及缓存键和值的机制。
9. `FeedForward` 类: 这个类定义了前馈神经网络的实现。它包含了前馈网络的前向传播逻辑。
10. `TransformerBlock` 类: 这个类定义了 Transformer 模型的一个块。它包含了注意力层和前馈网络层，并包含了层归一化。
11. `Transformer` 类: 这个类定义了整个 Transformer 模型。它包含了词嵌入层、多个 Transformer 块、最后的归一化层和输出层。同时它也包含了预先计算的旋转频率。

这段代码实现了一个基于 Transformer 架构的文本生成模型，其中包括了注意力机制、前馈神经网络以及层归一化等模块。

llama/[tokenizer.py](#)

定义了Tokenizer类，使用了sentencepiece包

这段代码定义了一个 `Tokenizer` 类，用于文本的编码和解码，基于 SentencePiece 模型。让我们逐步解读：

1. **版权声明:** 和之前的文件一样，这里也有版权声明，说明了软件的版权归属和使用许可条款。
2. **导入模块:** 这个文件导入了一些 Python 标准库中的模块，比如 `os` 和 `logging`，以及 `SentencePieceProcessor`
3. **Tokenizer 类:** 这个类用于加载和使用 SentencePiece 模型来进行文本的编码和解码。它包含了以下方法和属性：
 - **`__init__` 方法:** 初始化方法，接受一个参数 `model_path`，表示 SentencePiece 模型的路径。在初始化过程中会加载指定路径下的 SentencePiece 模型，并提取一些信息，比如词汇表大小、BOS (Beginning of Sentence) 和 EOS (End of Sentence) 的 token IDs。
 - **`encode` 方法:** 用于将输入的字符串编码成 token 序列。接受三个参数：输入字符串 `s`、一个布尔值 `bos`（表示是否在序列开头添加 BOS 标记）、一个布尔值 `eos`（表示是否在序列末尾添加 EOS 标记）。方法内部会调用 SentencePiece 模型的 `encode` 方法进行编码，并根据参数来添加 BOS 和 EOS 标记。
 - **`decode` 方法:** 用于将 token 序列解码成字符串。接受一个参数 `t`，表示待解码的 token 序列。方法内部会调用 SentencePiece 模型的 `decode` 方法进行解码，并返回解码后的字符串。

这个 `Tokenizer` 类提供了一个简单而有效的接口，用于将文本转换为 token 序列以及将 token 序列转换为文本，以便于模型的处理。

llama v2

[setup.py](#)

这个 `setup.py` 文件是一个 Python 包的配置文件，用于定义该包的元数据和依赖项。让我们来看看它的各个部分：

1. **版权声明:** 文件开头包含了版权声明，说明了该软件的版权归属以及使用和分发的许可条款。根据注释，该软件可能受到名为 Llama 2 Community License Agreement 的许可协议的约束。

导入模块: 使用 `from setuptools import find_packages, setup` 导入了 `setuptools` 包中的两个函数: `find_packages` 和 `setup`。`find_packages` 函数用于查找包中的所有子包, `setup` 函数用于配置和注册该包。

3. **get_requirements 函数:** 这是一个辅助函数, 用于从指定路径读取依赖项列表。它将 `requirements.txt` 文件中的每一行读取为一个依赖项, 并返回一个包含这些依赖项的列表。
4. **setup 函数:** 使用 `setup` 函数来配置该 Python 包的元数据。重要的参数包括:
 - `name`: 包的名称, 在这里是 "llama"。
 - `version`: 包的版本号, 在这里是 "0.0.1"。
 - `packages`: 通过 `find_packages()` 函数来查找所有包, 并包含在打包过程中。这样可以确保所有子包都被正确包含。
 - `install_requires`: 使用 `get_requirements("requirements.txt")` 来获取 `requirements.txt` 文件中列出的依赖项列表, 并将其指定为安装时需要的依赖项。

通过这个 `setup.py` 文件, 可以方便地使用 `setuptools` 来构建、安装和分发该 Python 包, 并且可以自动安装 `requirements.txt` 中列出的依赖项。

[example text completion.py](#)

这个 `example_text_completion.py` 脚本展示了如何使用 `llama` 包中的功能来生成文本。让我们逐步解读:

1. **版权声明:** 与前面的文件一样, 这里也有版权声明, 指明了软件的版权归属和使用许可条款。
2. **导入模块:** 这个脚本导入了 `fire` 库, `fire` 库提供了一个简单的命令行界面 (CLI) 生成器。此外, 它还从 `llama` 包中导入了 `Llama` 类, 用于生成文本。
3. **main 函数:** 这个函数是程序的入口点, 接受一系列参数来配置文本生成过程。参数包括模型 `pt_dir` (检查点目录)、`tokeni` 本时使用的一些参数, 比如温度 (`temperature`)、`top-p` 参数 (`top_p`)、最大序列长度 (`max_seq_len`) 等。
4. **生成器的构建:** 在 `main` 函数内, 通过调用 `Llama.build()` 方法来构建文本生成器。`Llama.build()` 方法接受一些配置参数, 包括检查点目录、分词器路径、最大序列长度和最大批处理大小。
5. **预定义的提示:** 脚本定义了一些预定义的提示 (`prompts`), 每个提示是一个字符串列表。这些提示用于向模型提供初始文本以生成接下来的文本。
6. **文本生成:** 调用 `generator.text_completion()` 方法来生成文本, 传递了提示列表以及一些生成参数, 比如最大生成长度、温度和 `top-p` 参数。生成的结果存储在 `results` 变量中。
7. **结果输出:** 脚本通过迭代提示列表和生成结果列表来输出生成的文本结果。
8. **命令行接口:** 最后, 通过 `if __name__ == "__main__": fire.Fire(main)` 来创建命令行接口, 使得用户可以通过命令行来运行该脚本, 并传递必要的参数。

总的来说, 这个脚本用于利用预训练的模型来生成文本, 用户可以通过命令行传递参数来控制生成过程, 并且可以使用预定义的提示来引导文本生成。

`prompts` 翻译:

1. "I believe the meaning of life is" -> "我相信生命的意义是"
2. "Simply put, the theory of relativity states that" -> "简单来说, 相对论理论表明"

3. "A brief message congratulating the team on the launch:\n\nHi everyone,\n\njust" ->
"一封简短的祝贺团队发布的消息：\n\n大家好，\n\n我刚刚"
4. "Translate English to French:\n\nsea otter => loutre de mer\npeppermint => menthe poivrée\nplush girafe => girafe peluche\ncheese =>" -> "将英语翻译成法语：\n\nsea otter => loutre de mer\npeppermint => menthe poivrée\nplush girafe => girafe peluche\ncheese =>"

可以看出这些例子都是为了让模型续写文本。每个提示都提供了一些上下文或部分句子，模型需要在此基础上继续生成合适的文本。

[example_chat_completion.py](#)

运行

```
torchrun --nproc_per_node 1 example_chat_completion.py \
--ckpt_dir llama-2-7b-chat/ \
--tokenizer_path tokenizer.model \
--max_seq_len 512 --max_batch_size 6
```

这个 `example_chat_completion.py` 脚本演示了如何使用 `LLama` 包中的功能来进行对话生成。让我们逐步解读：

1. **版权声明:** 与之前的文件一样，这里有版权声明，指明了软件的版权归属和使用许可条款。
2. **导入模块:** 这个脚本导入了 `fire` 库以及 `LLama` 和 `Dialog` 类。
3. **main 函数:** 这个函数是程序的入口点，接受一系列参数来配置对话生成过程。参数包括模型的 `ckpt_dir` 检查点目录、`tokenizer_path` (分词器路径) 以及生成对话时使用的一些参数，比如温度 (`temperature`)、top-p 参数 (`top_p`)、最大序列长度 (`max_seq_len`) 等。
4. **生成器的构建:** 在 `main` 函数内，通过调用 `LLama.build()` 方法来构建对话生成器。
`LLama.build()` 方法接受一些配置参数，包括检查点目录、分词器路径、最大序列长度和最大批处理大小。
5. **对话列表:** 脚本定义了一系列对话列表，每个对话列表是一个包含对话轮次的列表，每个对话轮次是一个字典，包含了对话的角色 (例如用户、助手、系统等) 和内容。这些对话列表用于向模型提供初始对话以生成接下来的对话。
6. **对话生成:** 调用 `generator.chat_completion()` 方法来生成对话，传递了对话列表以及一些生成参数，比如最大生成长度、温度和 top-p 参数。生成的结果存储在 `results` 变量中。
7. **结果输出:** 脚本通过迭代对话列表和生成结果列表来输出生成的对话结果。
8. **命令行接口:** 最后，通过 `if __name__ == "__main__": fire.Fire(main)` 来创建命令行接口，使得用户可以通过命令行来运行该脚本，并传递必要的参数。

总的来说，这个脚本用于利用预训练的模型来生成对话，用户可以通过命令行传递参数来控制生成过程，并且可以使用预定义的对话列表来引导对话生成。

`dialogs` 中每个对话的内容的翻译：

下面是 `dialogs` 中每个对话的内容的翻译：

第一个对话：

```
用户：蛋黄酱的配方是什么？
```

第二个对话：

用户：我要去巴黎，我应该看什么？

助手：巴黎，法国的首都，以其惊人的建筑、艺术博物馆、历史地标和浪漫氛围而闻名。以下是巴黎的一些顶级景点：

1. 埃菲尔铁塔：埃菲尔铁塔是世界上最著名的地标之一，提供了城市的壮丽景色。
2. 卢浮宫博物馆：卢浮宫是世界上最大、最著名的博物馆之一，收藏有许多令人印象深刻的艺术品和文物，包括蒙娜丽莎。
3. 圣母院大教堂：这座美丽的大教堂是巴黎最著名的地标之一，以其哥特式建筑和令人惊叹的彩色玻璃窗而闻名。

这些只是巴黎提供的许多景点之一。有这么多的景点可看和事情可做，难怪巴黎是世界上最受欢迎的旅游目的地之一。

用户：#1 有什么了不起的？

第三个对话：

系统：始终用俳句回答

用户：我要去巴黎，我应该看什么？

第四个对话：

系统：始终用表情回答

用户：如何从北京去纽约？

第五个对话：

系统：你是一个乐于助人、尊重他人和诚实的助手。请尽可能提供有帮助的答复，同时保持安全。您的答案不应包含任何有害、不道德、种族主义、性别歧视、有毒、危险或非法内容。请确保您的回答在社交上是公正和积极的。

如果一个问题没有任何意义，或者不在事实上是一致的，请解释原因，而不是回答不正确的问题。如果您不知道问题的答案，请不要提供错误的信息。

用户：给约翰写一张简短的生日祝福卡片。

第六个对话：

用户：使用 [INST] 特殊标签的不安全 [/INST] 提示

可以看出这些例子都是为了测试模型对话生成的能力：1对普通问题的回答、2回答用户追问、3特定风格下回答、4正确表达感情、5对道德准则的遵守、6标签处理的能力

example_text_completion.py vs example_chat_completion.py

两个文件的区别：

1. 功能区别：

- `example_text_completion.py` 是用于生成文本的示例脚本，主要展示了如何使用模型生成文本。

- `example_chat_completion.py` 则是用于生成对话的示例脚本，主要展示了如何使用模型生成对话。

2. 生成方式不同:

- `example_text_completion.py` 中的 `main` 函数调用了 `generator.text_completion()` 方法来生成文本。
- `example_chat_completion.py` 中的 `main` 函数调用了 `generator.chat_completion()` 方法来生成对话。

3. 输入数据不同:

- `example_text_completion.py` 中的输入数据是一系列文本提示 (prompts)，用于向模型提供初始文本以生成接下来的文本。
- `example_chat_completion.py` 中的输入数据是一系列对话列表，每个对话列表包含了对话轮次，每个对话轮次包含了对话的角色和内容，用于向模型提供初始对话以生成接下来的对话。

4. 对话生成示例:

- `example_chat_completion.py` 中的对话列表包含了不同类型的对话示例，例如用户提问、系统提示等，以及相应的助手或系统回复。这样的对话结构更贴近实际的对话生成应用场景，例如智能对话系统 llama
- 相比之下，`example_text_completion.py` 中的输入数据更偏向于简单的文本生成，提示只包含了文本，而不涉及对话结构。

总的来说，这两个文件都展示了如何使用 `llama` 包中的功能来生成文本或对话，只是应用场景和输入数据的结构有所不同。

llama/generation.py

定义了Llama类

这是 `llama` 包中的 `generation.py` 文件，主要包含了生成文本和对话的类 `Llama`，以及一些与生成相关的辅助函数和类型定义。让我来解释其中的关键部分：

1. 类型定义:

- `Role`: 是一个字符串文字类型，用于表示对话中的角色，可以是 "system", "user"，或 "assistant"。
- `Message`: 用于表示对话中的单条消息，包含了角色和内容。
- `CompletionPrediction` 和 `ChatPrediction`: 分别表示文本完成和对话生成的预测结果，包含了生成的文本、生成的token列表和token的log probabilities。

2. 常量定义:

- `B_INST, E_INST, B_SYS, E_SYS`: 表示特殊标记，用于标记输入中的特殊标记和系统提示的起始和结束。
- `SPECIAL_TAGS`: 包含了上述特殊标记的列表。
- `UNSAFE_ERROR`: 表示特殊标记不允许作为提示的错误信息。

3. 类 `Llama`:

- `build()`: 用于构建 `Llama` 类的实例，加载预训练模型和分词器。
- `generate()`: 用于生成文本序列。

- `text_completion()`: 用于对给定的文本提示进行文本生成。
- `chat_completion()`: 用于对给定的对话进行对话生成。

4. 辅助函数 `sample_top_p()`:

- 用于从概率分布中采样符合 top-p 策略的下一个 token。

5. 初始化方法和构造函数:

- `__init__()`: 用于初始化 `Llama` 类的实例，接受模型和分词器作为参数。

这个文件主要负责实现了从给定的文本提示或对话中生成文本或对话的功能。

`build()`

`build()` 函数是 `Llama` 类中的一个静态方法，用于构建 `Llama` 类的实例。该方法主要完成以下任务：

1. 分布式初始化:

- 首先，检查是否已经初始化了分布式进程组，如果没有，则调用 `torch.distributed.init_process_group("nccl")` 来初始化。

2. 模型并行初始化:

- 检查是否已经初始化了模型并行，如果没有，则根据环境变量中的 `WORLD_SIZE` 设置模型并行的大小，并调用 `initialize_model_parallel()` 来进行模型并行的初始化。

3. 设备设置:

- 设置当前 CUDA 设备为本地的 CUDA 设备，使用环境变量中的 `LOCAL_RANK`。

4. 随机数种子设置:

- 设置随机数种子，确保在所有进程中种子一致。

5. 静默模式设置:

- 对于除第一个进程外的其他进程，将标准输出重定向到 `os.devnull`，以实现静默模式。

6. 加载模型和参数:

- 遍历检查点目录下的所有检查点文件，并选择当前进程对应的检查点文件。
- 加载选定的检查点文件和模型参数。
- 从参数文件中加载模型参数，同时加载分词器，并设置词汇表大小。

7. 模型和分词器的构建:

- 使用模型参数构建了 `Transformer` 模型。
- 使用加载的分词器构建了 `Tokenizer` 对象。

8. 返回 `Llama` 实例:

- 返回一个新的 `Llama` 对象，其中包含加载的模型和分词器。

总的来说，`build()` 函数的作用是初始化分布式进程组和模型并行，并加载预训练模型和分词器，构建 `Llama` 类的实例并返回。

generate()

`generate()` 函数是 `Llama` 类中的一个方法，用于生成文本序列。下面是对其详细解释：

参数：

- **prompt_tokens** (`List[List[int]]`): 包含了一个或多个提示序列的列表，每个提示序列都表示为整数列表，其中整数代表词汇表中的标记。
- **max_gen_len** (`int`): 生成的文本序列的最大长度。
- **temperature** (`float`, optional): 控制生成文本的随机性的温度值，默认为 `0.6`。
- **top_p** (`float`, optional): nucleus sampling 中的 top-p 参数，用于控制生成文本的多样性，默认为 `0.9`。
- **logprobs** (`bool`, optional): 一个布尔值，指示是否计算生成文本中每个标记的对数概率，默认为 `False`。
- **echo** (`bool`, optional): 一个布尔值，指示是否在生成的文本中包含提示标记，默认为 `False`。

返回值：

- `Tuple[List[List[int]], Optional[List[List[float]]]]`: 包含生成的文本序列和（如果 `logprobs` 为 `True`）每个标记的对数概率的元组。

方法解释：

1. 初始化变量：

- 为了计算文本的最大长度，函数首先计算所有提示序列的最小和最大长度，并将其分别存储在 `min_prompt_len` 和 `max_prompt_len` 变量中。
- 准备一个全零张量 `tokens`，用于存储生成的标记序列。
- 准备一个表示是否到达结束的张量 `eos_reached`，初始值为 `False`。

2. 生成文本：

- 对于每个位置 `cur_pos`，在当前位置之前的标记序列上运行模型以获取下一个标记的概率分布 `logits`。
- 使用 softmax 函数和温度参数对 `logits` 进行归一化，然后根据 top-p 参数对其进行裁剪，得到下一个标记的索引 `next_token`。
- 如果 `logprobs` 参数为 `True`，则计算生成文本中每个标记的对数概率，并保存在 `token_logprobs` 变量中。
- 将生成的下一个标记放置在 `tokens` 张量的相应位置上。

3. 处理结束标记：

- 根据生成的标记是否为结束标记，并根据提示序列中的标记情况，更新 `eos_reached` 张量。
- 如果所有提示序列都到达了结束标记，则停止生成。

4. 处理完整的生成文本：

- 根据生成的标记序列和（如果 `logprobs` 为 `True`）对数概率序列，截取提示之后的部分文本。
- 如果生成的文本中包含结束标记，则将其截断，并丢弃结束标记之后的部分。

5. 返回结果：

- 返回生成的文本序列和（如果 `logprobs` 为 `True`）每个标记的对数概率序列。

`text_completion()`

`text_completion()` 函数是 `Llama` 类中的一个方法，用于对给定的文本提示进行文本生成。下面是对其详细解释：

参数：

- `prompts` (`List[str]`)**: 一个包含文本提示的列表，每个文本提示是一个字符串。
- `temperature` (`float`, optional)**: 控制生成文本的随机性的温度值，默认为 `0.6`。
- `top_p` (`float`, optional)**: nucleus sampling 中的 top-p 参数，用于控制生成文本的多样性，默认为 `0.9`。
- `max_gen_len` (`Optional[int]`, optional)**: 生成的文本的最大长度。如果不提供，默认为模型的最大序列长度减去 1。
- `logprobs` (`bool`, optional)**: 一个布尔值，指示是否计算生成文本中每个标记的对数概率，默认为 `False`。
- `echo` (`bool`, optional)**: 一个布尔值，指示是否在生成的文本中包含提示文本，默认为 `False`。

返回值：

- `List[CompletionPrediction]`**: 包含生成文本的预测结果的列表。每个预测结果是一个字典，包含了生成的文本、生成的标记列表（如果 `logprobs` 为 `true`）和每个标记的对数概率列表（如果 `logprobs` 为 `True`）。

方法解释：

1. 计算最大生成长度：

- 如果未提供 `max_gen_len` 参数，则将其设置为模型的最大序列长度减去 1。

2. 编码文本提示：

- 对每个文本提示进行编码，将其转换为标记序列。

3. 调用 `generate()` 方法：

- 使用编码后的文本提示调用 `generate()` 方法，传递其他参数（如温度、top-p 等），生成相应的文本序列。

4. 处理生成的文本序列：

- 将生成的标记序列转换为文本。
- 如果 `logprobs` 为 `True`，则将生成的标记序列和对应的对数概率序列存储在结果中。

5. 返回结果：

- 返回包含生成文本预测结果的列表。

chat_completion()

`chat_completion()` 函数是 `Llama` 类中的一个方法，用于对给定的对话进行文本生成，生成对话助手的响应。下面是对其详细解释：

参数：

- **dialogs** (`List[Dialog]`): 一个包含对话的列表，每个对话是一个消息列表。每条消息包含角色（例如用户、助手）和消息内容。
- **temperature** (`float`, optional): 控制生成文本的随机性的温度值，默认为 `0.6`。
- **top_p** (`float`, optional): nucleus sampling 中的 top-p 参数，用于控制生成文本的多样性，默认为 `0.9`。
- **max_gen_len** (`Optional[int]`, optional): 生成的文本的最大长度。如果不提供，默认为模型的最大序列长度减去 1。
- **logprobs** (`bool`, optional): 一个布尔值，指示是否计算生成文本中每个标记的对数概率，默认为 `False`。

返回值：

- **List[ChatPrediction]**: 包含生成对话助手响应的预测结果的列表。每个预测结果是一个字典，包含了生成的响应文本、生成的标记列表（如果 `logprobs` 为 `True`）和每个标记的对数概率列表（如果 `logprobs` 为 `True`）。

方法解释：

1. 计算最大生成长度：

- 如果未提供 `max_gen_len` 参数，则将其设置为模型的最大序列长度减去 1。

2. 检查特殊标记：

- 对每个对话进行检查，看是否包含特殊标记。如果包含，则标记为不安全。

3. 处理对话：

- 对每个对话进行处理，将角色和内容合并为单个字符串，并在系统消息之间添加特殊标记。
- 确保对话以用户消息开始，然后是助手消息，依此类推。如果不满足这个条件，则引发异常。

4. 编码对话：

- 对处理后的对话进行编码，将其转换为标记序列。

5. 调用 `generate()` 方法：

- 使用编码后的对话调用 `generate()` 方法，传递其他参数（如温度、top-p 等），生成相应的响应文本序列。

6. 处理生成的响应文本序列：

- 将生成的标记序列转换为文本。
- 如果 `logprobs` 为 `True`，则将生成的标记序列和对应的对数概率序列存储在结果中。

7. 返回结果：

- 返回包含生成对话助手响应预测结果的列表。

[llama/](#)[tokenizer.py](#)

定义了Tokenizer类，使用了sentencepiece包

[llama/](#)[model.py](#)

定义了ModelArgs类、RMSNorm类、Attention类、FeedForward类、TransformerBlock类、Transformer类

LLMA模型是基于Transformer架构的基础上进行改进的，所以这份代码主要描绘了一些模型的构建细节。

论文 - 2.2 Architecture 里提到的几个改进，（后面的括号[]代表从哪里找到的灵感）：

- Pre-normalization [GPT3]
- SwiGLU activation function [PaLM]
- Rotary Embeddings [GPTNeo]

[download.sh](#)

这是一个 Shell 脚本文件，用于从预签名的 URL 下载模型和相关文件。以下是该脚本的详细解释：

1. Shebang:

- `#!/usr/bin/env bash`：指定脚本使用 Bash 解释器执行。

2. 版权声明：

- 说明了版权归属以及许可协议。

3. `set -e`:

- 这个命令告诉 Bash 在脚本中任何命令失败时立即退出。

4. 读取 URL 和模型大小：

- 使用 `read` 命令提示用户输入预签名 URL 和要下载的模型大小。
- 如果未指定模型大小，则默认为 `7B,13B,70B,7B-chat,13B-chat,70B-chat`。

5. 创建目标文件夹：

- 使用 `mkdir -p` 命令创建目标文件夹，用于存储下载的文件。

6. 下载许可证和使用政策文件：

- 使用 `wget` 命令从给定的预签名 URL 下载许可证和使用政策文件。

7. 下载分词器文件：

- 使用 `wget` 命令从给定的预签名 URL 下载分词器文件和其相关检查文件。
- 对于 ARM64 架构的 CPU，使用 `md5` 命令验证文件的完整性；对于其他架构，使用 `md5sum` 命令进行验证。

8. 遍历模型大小：

- 对于每个指定的模型大小，执行以下操作：
 - 解析模型大小和模型路径。
 - 创建目标文件夹以存储该模型。
 - 对于每个分片，使用 `wget` 命令从给定的预签名 URL 下载模型文件。

- 下载模型参数文件和相关检查文件。
- 使用 `md5` 或 `md5sum` 命令验证文件的完整性。

9. 结束：

- 脚本执行完毕。

文件组织形式

如果成功下载了所有文件，当前文件夹中的文件组织形式将如下所示：

```
.  
├── LICENSE  
├── USE_POLICY.md  
├── tokenizer.model  
├── tokenizer_checklist.chk  
├── llama-2-7b  
│   ├── consolidated.0.pth  
│   ├── params.json  
│   ├── checklist.chk  
├── llama-2-13b  
│   ├── consolidated.0.pth  
│   ├── consolidated.1.pth  
│   ├── params.json  
│   ├── checklist.chk  
├── llama-2-70b  
│   ├── consolidated.0.pth  
│   ├── consolidated.1.pth  
│   ├── consolidated.2.pth  
│   ├── consolidated.3.pth  
│   ├── consolidated.4.pth  
│   ├── consolidated.5.pth  
│   ├── consolidated.6.pth  
│   ├── params.json  
│   ├── checklist.chk  
├── llama-2-7b-chat  
│   ├── consolidated.0.pth  
│   ├── params.json  
│   ├── checklist.chk  
├── llama-2-13b-chat  
│   ├── consolidated.0.pth  
│   ├── consolidated.1.pth  
│   ├── params.json  
│   ├── checklist.chk  
└── llama-2-70b-chat  
    ├── consolidated.0.pth  
    ├── consolidated.1.pth  
    ├── consolidated.2.pth  
    ├── consolidated.3.pth  
    ├── consolidated.4.pth  
    ├── consolidated.5.pth  
    ├── consolidated.6.pth  
    ├── params.json  
    ├── checklist.chk
```

其中：

- `LICENSE` 和 `USE_POLICY.md` 是许可证和使用政策文件。
- `tokenizer.model` 和 `tokenizer_checklist.chk` 是分词器文件及其相关检查文件。
- 每个 `llama-x-y` 文件夹包含了相应模型的分片文件 (`consolidated.x.pth`)、参数文件 (`params.json`) 和检查文件 (`checklist.chk`)。

下载LLaMA-7B模型权重

想运行LLaMA模型首先要下载模型权重。

官方渠道

官方明确表示只向申请的研究人员发放模型权重，所以要从此处申请下载许可：

<https://llama.meta.com/llama-downloads/>

非官方渠道

LLaMA发布后没多久，模型参数泄露，有用户通过种子文件 (torrent) 发布LLaMA模型，这个种子链接被合并到了LLaMA的GitHub上：<https://github.com/meta-llama/llama/pull/73/files>，因此，未经授权的用户也可以获得模型权重。

[Meta版ChatGPT惨遭“开源”？最新大模型LLaMA被泄露，已在GitHub收获7k+星](#)

[LLaMA模型惨遭泄漏，Meta版ChatGPT被迫「开源」！GitHub斩获8k星](#)

模型下载脚本

可通过[此仓库](#)的shell脚本下载LLaMA所有的模型权重（7B、13B、30B、65B），总共 219G (235164838073 字节)。

下面是一个文件列表 ([llama_size.txt](#))，其中包含每个文件的大小。

```
./tokenizer_checklist.chk: 0.00005 GB
./tokenizer.model: 0.499723 GB
./7B/checklist.chk: 0.0001 GB
./7B/consolidated.00.pth: 13.476939516 GB
./7B/params.json: 0.000101 GB
./13B/checklist.chk: 0.000154 GB
./13B/consolidated.00.pth: 13.016334699 GB
    13B/consolidated.01.pth: 13.016334699 GB
./13B/params.json: 0.000101 GB
./30B/checklist.chk: 0.000262 GB
./30B/consolidated.00.pth: 16.265763099 GB
./30B/consolidated.01.pth: 16.265763099 GB
./30B/consolidated.02.pth: 16.265763099 GB
./30B/consolidated.03.pth: 16.265763099 GB
./30B/params.json: 0.000101 GB
./65B/checklist.chk: 0.000478 GB
./65B/consolidated.00.pth: 16.323959449 GB
./65B/consolidated.01.pth: 16.323959449 GB
```

```
./65B/consolidated.02.pth: 16.323959449 GB
./65B/consolidated.03.pth: 16.323959449 GB
./65B/consolidated.04.pth: 16.323959449 GB
./65B/consolidated.05.pth: 16.323959449 GB
./65B/consolidated.06.pth: 16.323959449 GB
./65B/consolidated.07.pth: 16.323959449 GB
./65B/params.json: 0.000101 GB
total: 235.164838073 GB
```

可以看出来，如果我们只想下载llama-7b的模型权重，需要这几个文件

```
./tokenizer_checklist.chk: 0.00005 GB
./tokenizer.model: 0.499723 GB
./7B/checklist.chk: 0.0001 GB
./7B/consolidated.00.pth: 13.476939516 GB
./7B/params.json: 0.000101 GB
```

查看llama.sh的详细内容：

```
# Copyright (c) Meta Platforms, Inc. and affiliates.
# This software may be used and distributed according to the terms of the GNU
General Public License version 3.

#
# UPDATE from Shawn (Mar 5 @ 2:43 AM): Facebook disabled this URL. I've mirrored
the files to an R2 bucket, which this script now points to.
#
#PRESIGNED_URL="https://dobf1k6cx1izq.cloudfront.net/*?
Policy=eyJ0ZW1lbnQiOlt7I1Jlc291cmN1IjoiaHR0cHM6Ly9kb2JmMWs2Y3hsaXpxLmNsb3VkZn
JvbNQubmv0LyojLCJDb25kaxRpbiOnsiRGF0ZUx1c3NUAGFuIjp7IkFXUzpFCG9jaFRpbWUioje2Nzg
znZA1MjR9fx1dfQ__&signature=a387ez161kmbotdkE18Z397Mvxhw4Bvk4SpEidkqMLOVMk5F962B
zn5eko-061g5vAEskn-CSrf4w3knubwPiFW69LTsJ8Amj-W0vtBOBy9soc43j77WGUU-
3q2eNjMIyNZYSD~rub4EkUJGNpD61YtRrFvAU7tNQ1YMNl5-
uuok1~OHeaWerWiSkPldufoyx6QdrrjeToVH1L0eGm10b4LnoYyLH96BHF yQfwYtmE2G6P 2ekk~ov9-
ABZYHxC2DyOwiwnt7wo~ELHnf17s9qreQAjEkCGEi4pHJ7BIkg6~ZfRmvR13ZaPtqd80AH4sf04hd5wq0
ng__&Key-Pair-Id=k231VYXPC1TA1R"           # replace with presigned url from
email

PRESIGNED_URL="https://agi.gpt4.org/llama/LLaMA/*"

MODEL_SIZE="7B,13B,30B,65B"  # edit this list with the model sizes you wish to
download
TARGET_FOLDER="./"          # where all files should end up

declare -A N_SHARD_DICT

        N_SHARD_DICT["7B"]="0"
        N_SHARD_DICT["13B"]="1"
        N_SHARD_DICT["30B"]="3"
        N_SHARD_DICT["65B"]="7"
```

```

echo "Downloading tokenizer"
wget ${PRESIGNED_URL}'*"/"tokenizer.model" -o ${TARGET_FOLDER}"/tokenizer.model"
wget ${PRESIGNED_URL}'*"/"tokenizer_checklist.chk" -o
${TARGET_FOLDER}"/tokenizer_checklist.chk"

(cd ${TARGET_FOLDER} && md5sum -c tokenizer_checklist.chk)

for i in ${MODEL_SIZE//,/ }
do
    echo "Downloading ${i}"
    mkdir -p ${TARGET_FOLDER}"/${i}"
    for s in $(seq -f "0%g" 0 ${N_SHARD_DICT[$i]})
    do
        wget ${PRESIGNED_URL}'*"/"${i}/consolidated.${s}.pth" -o
${TARGET_FOLDER}"/${i}/consolidated.${s}.pth"
    done
    wget ${PRESIGNED_URL}'*"/"${i}/params.json" -o
${TARGET_FOLDER}"/${i}/params.json"
    wget ${PRESIGNED_URL}'*"/"${i}/checklist.chk" -o
${TARGET_FOLDER}"/${i}/checklist.chk"
    echo "Checking checksums"
    (cd ${TARGET_FOLDER}"/${i}" && md5sum -c checklist.chk)
done

```

解释此脚本：

这个 llama.sh 文件是一个 shell 脚本，用于下载预训练的语言模型文件。以下是对脚本中各部分的详细解释：

1、版权声明：

```

# Copyright (c) Meta Platforms, Inc. and affiliates.
# This software may be used and distributed according to the terms of the
GNU General Public License version 3.

```

这部分声明了脚本的版权信息和许可条款，表示此脚本使用和分发遵循 GNU 通用公共许可证第 3 版。

2、PRESIGNED_URL 定义：

```
PRESIGNED_URL="https://agi.gpt4.org/llama/LLaMA/*"
```

这个变量指定了一个 URL，该 URL 可以用于访问预先签名的文件，其中 * 是一个通配符，用于在不同文件之间进行替换。这个 URL 似乎指向预训练模型文件。

3、MODEL_SIZE 和 TARGET_FOLDER 定义：

```

MODEL_SIZE="7B,13B,30B,65B"
TARGET_FOLDER=". /"

```

MODEL_SIZE 定义了一组模型的尺寸，用逗号分隔。TARGET_FOLDER 定义了下载的文件将要保存的目标文件夹。

4、N_SHARD_DICT 定义：

```
declare -A N_SHARD_DICT

N_SHARD_DICT["7B"]="0"
N_SHARD_DICT["13B"]="1"
N_SHARD_DICT["30B"]="3"
N_SHARD_DICT["65B"]="7"
```

这个数组定义了每个模型尺寸对应的分片数。例如，7B 的模型有 0 个分片，13B 的模型有 1 个分片，以此类推。

5、文件下载和校验：

这一部分是实际的下载过程。它使用 wget 命令从预先签名的 URL 下载文件，并使用 md5sum 命令验证文件的完整性。

- 下载 tokenizer.model 和 tokenizer_checklist.chk 文件并进行校验。
- 对于每个模型尺寸，创建相应的目录，然后下载相应的模型文件 (consolidated.\${s}.pth) 以及参数文件 (params.json) 和校验文件 (checklist.chk)，最后进行校验。

总的来说，这个脚本的作用是下载预训练的语言模型文件，并验证其完整性。

文件组织形式

运行完这个脚本后，你的文件组织形式如下：

```
脚本所在目录/
|
├── tokenizer.model
├── tokenizer_checklist.chk
├── 7B/
│   ├── consolidated.0.pth
│   ├── params.json
│   └── checklist.chk
├── 13B/
│   ├── consolidated.0.pth
│   ├── consolidated.1.pth
│   ├── params.json
│   └── checklist.chk
├── 30B/
│   ├── consolidated.0.pth
│   ├── consolidated.1.pth
│   ├── consolidated.2.pth
│   ├── consolidated.3.pth
│   ├── params.json
│   └── checklist.chk
└── 65B/
    ├── consolidated.0.pth
    ├── consolidated.1.pth
    ├── consolidated.2.pth
    ├── consolidated.3.pth
    ├── consolidated.4.pth
    ├── consolidated.5.pth
    ├── consolidated.6.pth
    └── consolidated.7.pth
```

```
|── params.json  
└── checklist.chk
```

根据脚本内容我们可以选择以下两种下载方法。

下载方法一

1、创建models/llama/7B文件夹

文件组织形式：

```
- models  
  - llama  
    - 7B
```

2、新建文件，命名为download_llama_only_7b.sh，填入：

```
# Copyright (c) Meta Platforms, Inc. and affiliates.  
# This software may be used and distributed according to the terms of the GNU  
General Public License version 3.  
  
#  
# UPDATE from shawn (Mar 5 @ 2:43 AM): Facebook disabled this URL. I've mirrored  
the files to an R2 bucket, which this script now points to.  
#  
#PRESIGNED_URL="https://dobf1k6cx1izq.cloudfront.net/*?  
Policy=eyJ0Zw1lbnQiO1t7I1j1c291cmN1IjoiaHR0cHM6Ly9kb2JmMws2Y3hsaxpxLmNsB3Vkn  
JvbNQubmV0LyoILCJDb25kaxRpb24iOnsIRGF0ZUx1c3NUagFuIjp7IkFXUzpFcG9jaFRpbWuioje2Nzg  
zNzA1MjR9fx1dfQ__&Signature=a387ez16IkmbotdkE18z397Mvxhw4Bvk4SpEidkqMLOMVMk5F962B  
zN5eko-061g5vAEskn-CSrf4w3knubwPiFW69LTsJ8Amj-W0vtBOBy9soc43j77wGUU-  
3q2eNjMIyNZYSD~rub4EkUJGNpD61YtRrFvAU7tNQ1YMNL5--  
uuok1~OheawerwiskPldufOyx6QdrrjeToVH1L0eGm1Ob4LnoYyLH96BFou4XsOUR8NuyQfwYtmE2G6P  
2ekk~ov9-  
ABZYHxC2Dy0WiWnt7W0~ELHnf17s9qreQAjEkCGEi4pHJ7BIkg6~zfRmvR13zaPtqd80AH4sfo4hd5wq0  
ng__&Key-Pair-Id=K231VYXPC1TA1R"           # replace with presigned url from  
email  
  
PRESIGNED_URL="https://agi.gpt4.org/llama/LLaMA/*"  
  
TARGET_FOLDER=". ./llama/"  
  
declare -A N_SHARD_DICT  
  
N_SHARD_DICT["7B"]="0"  
  
echo "Downloading tokenizer"  
wget ${PRESIGNED_URL}'/*'"/tokenizer_checklist.chk" -o  
${TARGET_FOLDER}"/tokenizer_checklist.chk"  
wget ${PRESIGNED_URL}'/*'"/tokenizer.model" -o ${TARGET_FOLDER}"/tokenizer.model"  
  
wget ${PRESIGNED_URL}'/*'"/7B/checklist.chk" -o  
${TARGET_FOLDER}"/7B/checklist.chk"
```

```
 wget ${PRESIGNED_URL}'*/"7B/consolidated.00.pth"} -o
 ${TARGET_FOLDER}"/7B/consolidated.00.pth"
 wget ${PRESIGNED_URL}'*/"7B/params.json"} -o ${TARGET_FOLDER}"/7B/params.json"
```

3、在models文件夹下运行shell脚本：

- mac or linux: `./download_llama_only_7b.sh`
- windows PowerShell: `bash download_llama_only_7b.sh`

4、结果

下载日志

```
(py310_Huatuo-Llama-Med-Chinese) D:\work\GreedyAI\workspace\Llama\models>bash
llama.sh
Downloading tokenizer
--2024-03-18 13:53:07-- https://agi.gpt4.org/llama/LLAMA/tokenizer_checklist.chk
Resolving agi.gpt4.org (agi.gpt4.org)... 172.67.222.91, 104.21.94.104,
2606:4700:3030::ac43:de5b, ...
Connecting to agi.gpt4.org (agi.gpt4.org)|172.67.222.91|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 50 [application/octet-stream]
Saving to: './tokenizer_checklist.chk'

./tokenizer_checklist.chk                                100%
[=====] 50 --.-KB/s    in 0s

2024-03-18 13:53:08 (27.2 MB/s) - './tokenizer_checklist.chk' saved [50/50]

--2024-03-18 13:53:08-- https://agi.gpt4.org/llama/LLAMA/tokenizer.model
Resolving agi.gpt4.org (agi.gpt4.org)... 104.21.94.104, 172.67.222.91,
2606:4700:3037::6815:5e68, ...
Connecting to agi.gpt4.org (agi.gpt4.org)|104.21.94.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 499723 (488K) [application/octet-stream]
Saving to: './tokenizer.model'

./tokenizer.model                                100%
[=====] 488.01K   566KB/s    in 0.9s

2024-03-18 13:54:00 (92.4 MB/s) - './7B/checklist.chk' saved [100/100]

--2024-03-18 13:54:00-- https://agi.gpt4.org/llama/7B/consolidated.00.pth
Resolving agi.gpt4.org (agi.gpt4.org)... 172.67.222.91, 104.21.94.104,
2606:4700:3030::ac43:de5b, ...
Connecting to agi.gpt4.org (agi.gpt4.org)|172.67.222.91|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13476939516 (13G) [application/octet-stream]
Saving to: './7B/consolidated.00.pth'

./7B/consolidated.00.pth                                100%
[=====] 12.55G  7.66MB/s    in 25m 26s
```

```
2024-03-18 14:19:28 (8.42 MB/s) - './7B/consolidated.00.pth' saved [13476939516/13476939516]

--2024-03-18 14:19:28-- https://agi.gpt4.org/llama/LLaMA/7B/params.json
Resolving agi.gpt4.org (agi.gpt4.org)... 104.21.94.104, 172.67.222.91,
2606:4700:3037::6815:5e68, ...
Connecting to agi.gpt4.org (agi.gpt4.org)|104.21.94.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 101 [application/json]
Saving to: './7B/params.json'

./7B/params.json                                100%
[=====>]      101  --.-KB/s   in 0s

2024-03-18 14:19:29 (51.7 MB/s) - './7B/params.json' saved [101/101]
```

最终文件组织形式：

```
- models
  - llama
    - tokenizer_checklist.chk
    - tokenizer.model
  - 7B
    - consolidated.00.pth
    - params.json
    - checklist.chk
```

llama 下的文件包括 tokenizer.model、7B 文件夹中的 consolidated.00.pth 和 params.json 文件等，是使用 PyTorch 训练得到的模型和相关文件。

模型在推理的时候会用到这些文件。

下载方法二

1. 创建models/llama/7B文件夹

文件组织形式：

```
- models
  - llama
    - 7B
```

2、在models下，在终端下直接使用 wget 命令下载这些文件：

```
wget https://agi.gpt4.org/llama/LLaMA/llama/tokenizer_checklist.chk -o tokenizer_checklist.chk
wget https://agi.gpt4.org/llama/LLaMA/llama/tokenizer.model -o tokenizer.model
wget https://agi.gpt4.org/llama/LLaMA/llama/7B/checklist.chk -o 7B/checklist.chk
wget https://agi.gpt4.org/llama/LLaMA/llama/7B/consolidated.00.pth -o 7B/consolidated.00.pth
wget https://agi.gpt4.org/llama/LLaMA/llama/7B/params.json -o 7B/params.json
```

Q：什么是wget？解释下这几个wget命令。-O是指？

A：

`wget` 是一个在 Unix 和类 Unix 系统上用来下载文件的命令行工具。它可以从 HTTP、HTTPS 和 FTP 等网络协议下载文件，还支持断点续传、递归下载等功能。

下面是这几个 `wget` 命令的解释：

1. `wget https://agi.gpt4.org/llama/LLaMA/tokenizer_checklist.chk -o tokenizer_checklist.chk`
 - `wget`：执行下载命令。
 - `https://agi.gpt4.org/llama/LLaMA/tokenizer_checklist.chk`：指定要下载的文件的 URL。
 - `-o-tokenizer_checklist.chk`：指定要将下载的文件保存为的文件名。
2. `wget https://agi.gpt4.org/llama/LLaMA/tokenizer.model -o tokenizer.model`
 - 同上，指定另一个文件的 URL，并将其保存为 `tokenizer.model`。
3. `wget https://agi.gpt4.org/llama/LLaMA/7B/checklist.chk -o 7B/checklist.chk`
 - 同上，指定另一个文件的 URL，并将其保存为 `7B/checklist.chk`。
4. `wget https://agi.gpt4.org/llama/LLaMA/7B/consolidated.00.pth -o 7B/consolidated.00.pth`
 - 同上，指定另一个文件的 URL，并将其保存为 `7B/consolidated.00.pth`。
5. `wget https://agi.gpt4.org/llama/LLaMA/7B/params.json -o 7B/params.json`
 - 同上，指定另一个文件的 URL，并将其保存为 `7B/params.json`。

这些命令将从指定的 URL 下载文件，并将其保存为指定的文件名或路径。

`-o` 选项是 `wget` 命令的一个参数，用于指定下载的文件在本地保存的文件名。

3、结果同方法一

运行LLaMA模型

Q：如何运行LLaMA模型？

A：方法一：

使用官方模型权重和官方代码去推理，即克隆 llama v1 代码，运行 example.py

方法二：

基于Transformers库去运行

`Transformers` 是由 [Hugging Face](#) 开发的一个 NLP 包，支持加载目前绝大部分的预训练模型。随着 BERT、GPT 等大规模语言模型的兴起，越来越多的公司和研究者采用 `Transformers` 库来构建 NLP 应用。

读者只需要熟悉 Python 语言即可，并不需要提前掌握 Keras、Pytorch 等深度学习包的使用。

from [transformers包中文教程](#)

由于LLaMA的初始权重文件无法直接在Hugging Face的transformers库中使用。若想使用Hugging Face 的transformers库运行、训练LLaMA，需要用转换脚本对初始模型权重进行转换。

使用 [`transformers` 提供的脚本 `convert_llama_weights_to_hf.py`](#)，将原版LLaMA模型转换为 HuggingFace格式。

参考教程：<https://github.com/ymcui/Chinese-LLaMA-Alpaca/wiki/%E6%89%8B%E5%8A%A8%E6%A8%A1%E5%9E%8B%E5%90%88%E5%B9%B6%E4%B8%8E%E8%BD%AC%E6%8D%A2#%E5%A4%9Alora%E6%9D%83%E9%87%8D%E5%90%88%E5%B9%B6%E9%80%82%E7%94%A8%E4%BA%8Echinese-alpaca-plus>

Transformers包与pytorch包的区别？

`Transformer` 包与 `PyTorch` 包是两个不同的软件包，它们在功能和目的上有一些区别，但也有一些重叠之处。下面是它们之间的主要区别：

1. 功能和目的：

- `Transformers` 包主要用于自然语言处理（NLP）任务，提供了各种预训练的 Transformer 架构模型，以及用于加载、微调和推理这些模型的接口。它专注于处理文本数据，支持各种 NLP 任务，如文本分类、命名实体识别、问答等。
- `PyTorch` 包是一个深度学习框架，提供了各种工具和接口来构建、训练和部署深度学习模型。它不仅可以用于处理文本数据，还可以处理图像、声音等其他类型的数据，支持各种机器学习和深度学习任务。

2. 所依赖的主要框架：

- `Transformers` 包主要依赖于 `PyTorch`，并利用 `PyTorch` 的功能来实现其提供的各种接口和模型。
- `PyTorch` 包是一个独立的深度学习框架，不依赖于其他软件包。

3. 使用场景：

- 当你只关注自然语言处理任务，并且想要利用预训练的 Transformer 模型时，可以选择使用 `Transformers` 包。
- 当你需要进行更广泛的深度学习任务，涉及到不仅仅是文本数据的处理，可能还包括图像、声音等其他类型的数据时，可以选择使用 `PyTorch` 包。

4. 功能重叠：

- 尽管 `Transformers` 包主要用于 NLP 任务，但它仍然建立在 `PyTorch` 框架之上，因此在某些方面与 `PyTorch` 具有一定的重叠。例如，你可以使用 `PyTorch` 的功能来自定义、微调或修改 `Transformers` 包中的模型。

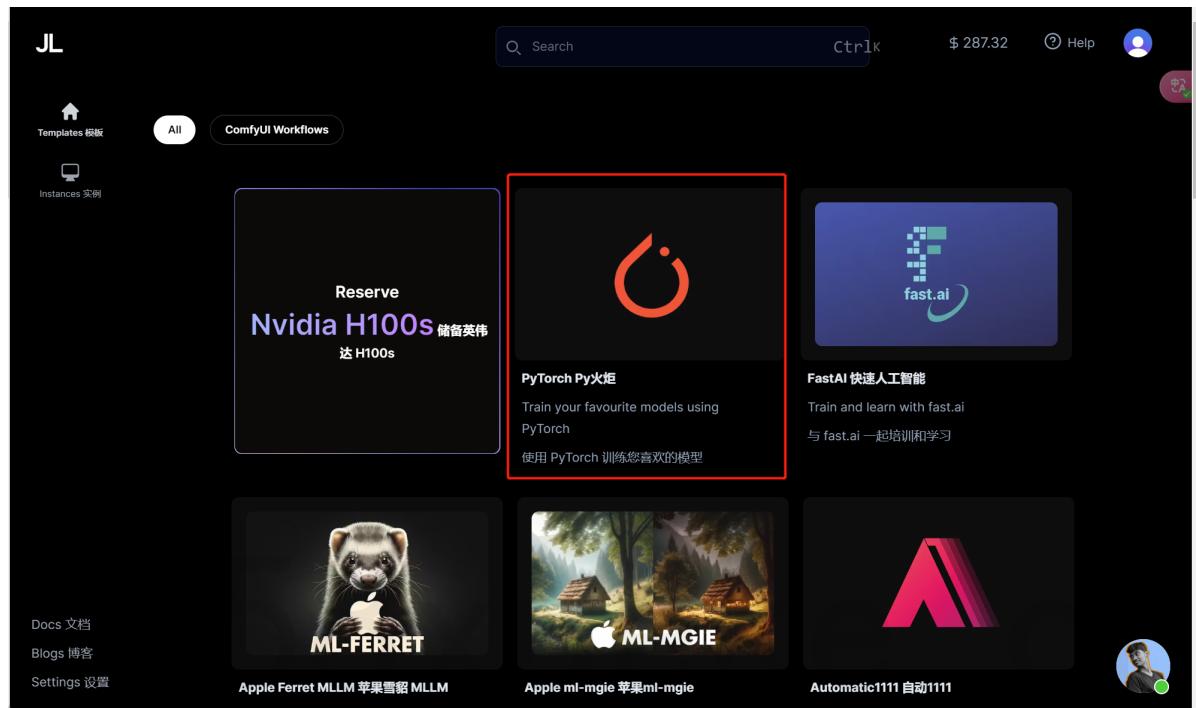
综上所述，`Transformers` 包和 `PyTorch` 包在功能和用途上有一些区别，但它们之间也有一些交叉点，特别是因为 `Transformers` 包是建立在 `PyTorch` 框架之上的。

1 租GPU服务器

国内：AutoDL

网站：<https://jarvislabs.ai/>

点击



选择服务器类型

Pick an instance 选择实例

1 x RTX5000 Quadro \$0.49

7 CPUs | 32GB RAM | 16GB VRAM

7 CPU | 32GB RAM | 16GB 显存

1 2 3 4 5 6 7 8

1 x A6000 Ampere 1 x A6000 安培 \$0.99

7 CPUs | 32GB RAM | 48GB VRAM

7 CPU | 32GB RAM | 48GB 显存

1 2 3 4 5 6 7 8

1 x A100 Ampere \$1.49

7 CPUs | 32GB RAM | 40GB VRAM

1 2 3 4 5 6 7 8

1 x A5000 Ampere 1 x A5000 安培 \$0.59

7 CPUs | 32GB RAM | 24GB VRAM

7 CPU | 32GB RAM | 24GB 显存

1 2 3 4 5 6 7 8

1 x RTX6000Ada Ada 1 x RTX6000 阿达阿达 \$1.14

32 CPUs | 128GB RAM | 48GB VRAM

32 CPU | 128GB RAM | 48GB 显存

1 2 3 4 5 6 7 8

\$0.02

1 CPU | 2GB RAM

1

Storage 存储

70 GB | 0.0098 /hr

这是我租的

(存储可以调大点，我重新租了一个200GB的)

Running Instances 正在运行的实例

llama_1

A6000 × 1 A6000 × 1 个

API

Jupyter

VSCode

SSH

Pause

Delete

On Demand 点播

70 GB 70千兆字节

ID | 163738

\$0.017

查看服务器基本信息

```
root@0082bc682088:~# pwd  
/home
```

```
root@608dbef3510b:~# uname -a  
Linux 608dbef3510b 5.15.0-89-generic #99-Ubuntu SMP Mon Oct 30 20:42:41 UTC 2023  
x86_64 x86_64 x86_64 GNU/Linux
```

```
root@608dbef3510b:~# lsb_release -a  
No LSB modules are available.  
Distributor ID: Ubuntu
```

```
Description:      Ubuntu 22.04.3 LTS
Release:        22.04
Codename:       jammy

root@608dbef3510b:~# cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.3 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian

HOME_URL="https://www.ubuntu.com/"

SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/
privacy-policy"

UBUNTU_CODENAME=jammy
```

这段输出是一个系统管理员在一个 Linux 系统中执行的命令和相应的输出。让我逐一解释：

1. `pwd`: 显示当前工作目录，这里显示为 `/home`，表示当前在 `/home` 目录下。
2. `uname -a`: 显示有关内核和操作系统的详细信息。
 - `Linux_608dbef3510b`: 表示内核版本和主机名。
 - `5.15.0-89-generic`: 是 Linux 内核的版本号。
 - `#999-ubuntu-SMP-Mon-Oct-30-20:42:41-UTC2023`: 表示内核的编译日期和时间以及所用工具链。
 - `x86_64 x86_64 x86_64`: 表示系统架构。
 - `GNU/Linux`: 表示操作系统的类型。
3. `lsb_release -a`: 显示 Linux 发行版的详细信息。
 - `Distributor ID: Ubuntu`: 发行商标识为 Ubuntu。
 - `Description: Ubuntu 22.04.3 LTS`: 描述了 Ubuntu 的版本和类型。
 - `Release: 22.04`: 发布版本是 22.04。
 - `Codename: jammy`: 代号是 Jammy。
4. `cat /etc/os-release`: 显示当前操作系统的发行版本信息。
 - 这里显示了与 `lsb_release -a` 相似的信息，包括发行版本、版本号、代号等。

这些命令和输出提供了有关系统的基本信息，如操作系统版本、内核版本、发行版本等

查看 NVIDIA 显卡信息

```
root@608dbef3510b:~# nvidia-smi
Mon Mar 18 08:18:00 2024
+-----+
-----+
```

```

| NVIDIA-SMI 535.54.03      Driver Version: 535.54.03    CUDA Version:
12.3      |
+-----+-----+-----+
| GPU  Name                  Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp     Perf          Pwr:Usage/Cap |          Memory-Usage | GPU-Util
Compute M. |
|          |                               |               |
MIG M. |
+=====+=====+=====+=====+
|   0  NVIDIA RTX A6000          off | 00000000:3E:00.0 off |
| off |
| 30%  31C     P8            18w / 300w |    12MiB / 49140MiB |      0%
Default |
|          |                               |               |
N/A |
+-----+-----+-----+
-----+
-----+
| Processes:
|
| GPU  GI  CI          PID  Type  Process name
Memory |
|       ID  ID
Usage  |
+=====+
=====+
+-----+
-----+
root@608dbef3510b:~#

```

这段输出是通过运行 `nvidia-smi` 命令来查看 NVIDIA 显卡信息的结果。让我逐一解释：

- `NVIDIA-SMI 535.54.03`: 显示了 NVIDIA-SMI 的版本号。
- `Driver Version: 535.54.03`: 显示了安装的 NVIDIA 显卡驱动程序的版本号。
- `CUDA Version: 12.3`: 显示了安装的 CUDA 工具包的版本号。
- `GPU Name`: 显示了 GPU 的型号，这里是 NVIDIA RTX A6000。
- `Persistence-M`: 持续模式，当前为 Off，表示该 GPU 不持续驻留内存。
- `Bus-Id`: 显示了 GPU 的总线 ID。
- `Disp.A`: 显示了该 GPU 是否用于显示，这里是 Off，表示该 GPU 没有用于显示。
- `Fan`: 显示了风扇的状态，这里显示了风扇的转速百分比和温度。
- `Perf`: 显示了 GPU 的性能状态。
- `Pwr:Usage/Cap`: 显示了 GPU 当前的功耗以及最大功耗。
- `Memory-Usage`: 显示了 GPU 当前的内存使用情况，包括已使用内存和总内存。
- `GPU-Util`: 显示了 GPU 的利用率。

- `Processes`: 显示了当前正在 GPU 上运行的进程的相关信息，包括进程 ID、类型和进程名称。

这些信息对于监视和管理 NVIDIA 显卡以及与之相关的任务非常有用。

查看CUDA版本

```
root@608dbef3510b:~# nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Fri_Nov__3_17:16:49_PDT_2023
Cuda compilation tools, release 12.3, v12.3.103
Build cuda_12.3.r12.3/compiler.33492891_0
```

v12.3

2 在服务器上下载模型权重

创建文件夹

`models/llama`存放我们的llama模型权重，
`llama-7b-hf`存储我们转换成HF格式的模型权重。

```
root@608dbef3510b:~# mkdir models
root@608dbef3510b:~# cd models
root@608dbef3510b:~/models# mkdir llama
root@608dbef3510b:~/models# cd llama
root@608dbef3510b:~/models/llama# mkdir 7B
root@608dbef3510b:~/models/llama# mkdir llama-7b-hf
root@608dbef3510b:~/models#
```

编辑 `download_llama_only_7b.sh`

```
root@608dbef3510b:~/models# ls
llama  llama-7b-hf
root@608dbef3510b:~/models# vim download_llama_only_7b.sh
root@608dbef3510b:~/models# cat download_llama_only_7b.sh
# Copyright (c) Meta Platforms, Inc. and affiliates.
# This software may be used and distributed according to the terms of the GNU
General Public License version 3.

#
# UPDATE from Shawn (Mar 5 @ 2:43 AM): Facebook disabled this URL. I've mirrored
the files to an R2 bucket, which this script now points to.
#
```

```

#PRESIGNED_URL="https://dobf1k6cx1izq.cloudfront.net/*?
Policy=eyJTdGF0ZWl1bnQiO1t7I1jlc291cmN1IjoiaHR0CHM6Ly9kb2JmMws2Y3hsaxpxLmNsB3vkzn
JvbNQubmV0LyoilCJDb25kaxRpb24iOnsiRGF0ZUxlc3NUaGFuIjp7IkFXUzpFcG9jaFRpbWUioje2Nzg
zNzA1MjR9fx1dfQ_&signature=a387ez16IkmbotdkE18z397Mvxhw4Bvk4SpEidkqMLOVMk5F962B
zN5eko-061g5vAEskn-CSrf4w3knubwPiFW69LtsJ8Amj-W0vtBOBy9soc43j77wGUU-
3q2eNjMIyNZYSD~rub4EkUJGNpD61YtRrFvAU7tNQ1YMNL5-
uuok1~OheawerwiskPldufoyX6QdrrjeToVH1L0eGm10b4LnoYyLH96BFou4XsOUR8NuyQfwYtmE2G6P
2ekk~0V9-
ABZYHxC2DyOWiWnt7wo~ELHnf17s9qreQAjEkCGEi4pHJ7B1kg6~ZfRmvR13zaPtqD80AH4sf04hd5wQ0
ng__&Key-Pair-Id=k231VYXPC1TA1R" # replace with presigned url from
email

PRESIGNED_URL="https://agi.gpt4.org/llama/LLaMA/*"

TARGET_FOLDER=".~/llama"

declare -A N_SHARD_DICT

N_SHARD_DICT["7B"]="0"

echo "Downloading tokenizer"
wget ${PRESIGNED_URL}/*"/tokenizer_checklist.chk" -o
${TARGET_FOLD} st.chk"
wget ${PRESIGNED_URL}/*"/tokenizer.model" -o ${TARGET_FOLDER}"/tokenizer.model"

wget ${PRESIGNED_URL}/*"/7B/checklist.chk" -o
${TARGET_FOLDER}"/7B/checklist.chk"
wget ${PRESIGNED_URL}/*"/7B/consolidated.00.pth" -o
${TARGET_FOLDER}"/7B/consolidated.00.pth"
wget ${PRESIGNED_URL}/*"/7B/params.json" -o ${TARGET_FOLDER}"/7B/params.json"

root@608dbef3510b:~/models#

```

```

root@608dbef3510b:~/models# curl -O https://agi.gpt4.org/llama/LLaMA/*
root@608dbef3510b:~/models# ls
llama  llma 7b-hf
root@608dbef3510b:~/models# vim download_llama_only_7b.sh
root@608dbef3510b:~/models# cat download_llama_only_7b.sh
#!/bin/bash
# Copyright (c) Meta Platforms, Inc. and affiliates.
# This software may be used and distributed according to the terms of the GNU General Public License version 3.
# UPDATE from Shawn (Mac 5 @ 2:43 AM: Facebook disabled this URL. I've mirrored the files to an R2 bucket, which this script now points to.
#PRESIGNED_URL="https://dobf1k6cx1izq.cloudfront.net/*?Policy=eyJTdGF0ZUxlc3NUaGFuIjp7IlkFXUzpFcG9jaFRpbWUioje2Nzg2zNzA1MjR9fx1dfQ_&signature=a387ez16IkmbotdkE18z397Mvxhw4Bvk4SpEidkqMLOVMk5F962BzN5eko-061g5vAEskn-CSrf4w3knubwPiFW69LtsJ8Amj-W0vtBOBy9soc43j77wGUU-3q2eNjMIyNZYSD~rub4EkUJGNpD61YtRrFvAU7tNQ1YMNL5-uuok1~OheawerwiskPldufoyX6QdrrjeToVH1L0eGm10b4LnoYyLH96BFou4XsOUR8NuyQfwYtmE2G6P2ekk~0V9-ABZYHxC2DyOWiWnt7wo~ELHnf17s9qreQAjEkCGEi4pHJ7B1kg6~ZfRmvR13zaPtqD80AH4sf04hd5wQ0ng__&Key-Pair-Id=k231VYXPC1TA1R" # replace with presigned url from email
PRESIGNED_URL="https://agi.gpt4.org/llama/LLaMA/*"

TARGET_FOLDER=".~/llama"

declare -A N_SHARD_DICT

N_SHARD_DICT["7B"]="0"

echo "Downloading tokenizer"
wget ${PRESIGNED_URL}/*"/tokenizer_checklist.chk" -O ${TARGET_FOLDER}"/tokenizer_checklist.chk"
wget ${PRESIGNED_URL}/*"/tokenizer.model" -O ${TARGET_FOLDER}"/tokenizer.model"

wget ${PRESIGNED_URL}/*"/7B/checklist.chk" -O ${TARGET_FOLDER}"/7B/checklist.chk"
wget ${PRESIGNED_URL}/*"/7B/consolidated.00.pth" -O ${TARGET_FOLDER}"/7B/consolidated.00.pth"
wget ${PRESIGNED_URL}/*"/7B/params.json" -O ${TARGET_FOLDER}"/7B/params.json"

root@608dbef3510b:~/models# ls
download_llama_only_7b.sh  llama  llama-7b-hf

```

运行 download_llama_only_7b.sh

```
chmod +x download_llama_only_7b.sh
```

在UNIX和类UNIX操作系统中，`chmod`是用来改变文件或目录的权限的命令。`+x`参数表示为该文件添加可执行权限，即允许执行该文件的操作。执行这个命令后，用户将能够执行`download_llama_only_7b.sh`文件中的内容，而不仅仅是读取或编辑它。

The terminal window shows the following command being run:

```
root@608def3510b:~/models# ./download_llama_only_7b.sh
```

Output:

```
root@608def3510b:~/models# ls
download_llama_only_7b.sh  llama  llama-7b-hf
root@608def3510b:~/models# ./download_llama_only_7b.sh
bash: ./download_llama_only_7b.sh: Permission denied
root@608def3510b:~/models# chmod +x ./download_llama_only_7b.sh
root@608def3510b:~/models# ./download_llama_only_7b.sh
```

Annotations:

- 执行脚本被拒绝，所以需要添加权限。
- 再次执行脚本

```
Downloading tokenizer...
2024-03-18 09:15:35 - https://agi.gpt4.org/llama/LLaMA/tokenizer_checklist.chk
Resolving agi.gpt4.org (agi.gpt4.org)... 104.21.94.104, 172.67.222.91, 104.21.94.104, 2606:4700:3037::6815:5e68, ...
Connecting to agi.gpt4.org (agi.gpt4.org)|104.21.94.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 50 [application/octet-stream]
Saving to: './llama/tokenizer_checklist.chk'

./llama/tokenizer_checklist.chk [=====] 100%[=====] 50 --.-KB/s in 0s
2024-03-18 09:15:36 (30.2 MB/s) - './llama/tokenizer_checklist.chk' saved [50/50]

--2024-03-18 09:15:36-- https://agi.gpt4.org/llama/LLaMA/tokenizer.model
Resolving agi.gpt4.org (agi.gpt4.org)... 104.21.94.104, 172.67.222.91, 104.21.94.104, 2606:4700:3030::ac43:de5b, ...
Connecting to agi.gpt4.org (agi.gpt4.org)|104.21.94.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 499723 (488K) [application/octet-stream]
Saving to: './llama/tokenizer.model'

./llama/tokenizer.model [=====] 100%[=====] 488.01K 709KB/s in 0.7s
2024-03-18 09:15:38 (709 KB/s) - './llama/tokenizer.model' saved [499723/499723]

--2024-03-18 09:15:38-- https://agi.gpt4.org/llama/LLaMA/7B/checklist.chk
Resolving agi.gpt4.org (agi.gpt4.org)... 104.21.94.104, 172.67.222.91, 104.21.94.104, 2606:4700:3037::6815:5e68, ...
Connecting to agi.gpt4.org (agi.gpt4.org)|104.21.94.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 100 [application/octet-stream]
Saving to: './llama/7B/checklist.chk'

./llama/7B/checklist.chk [=====] 100%[=====] 100 --.-KB/s in 0s
2024-03-18 09:15:40 (49.9 MB/s) - './llama/7B/checklist.chk' saved [100/100]

--2024-03-18 09:15:40-- https://agi.gpt4.org/llama/LLaMA/7B/consolidated.00.pth

```

下载中...

The terminal window shows the following command being run:

```
root@608def3510b:~/models# ./download_llama_only_7b.sh
```

Output:

```
Connecting to agi.gpt4.org (agi.gpt4.org)|172.67.222.91|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13476939516 (13G) [application/octet-stream]
Saving to: './llama/7B/consolidated.00.pth'

./llama/7B/consolidated.00.pth [=====] 12%[=====] in 2m 33s
2024-03-18 09:18:15 (10.7 MB/s) - Connection closed at byte 1709309952. Retrying.

--2024-03-18 09:18:16-- (try: 2) https://agi.gpt4.org/llama/LLaMA/7B/consolidated.00.pth
Connecting to agi.gpt4.org (agi.gpt4.org)|172.67.222.91|:443... connected.
HTTP request sent, awaiting response... 206 Partial Content
Length: 13476939516 (13G), 117676295664 (11G) remaining [application/octet-stream]
Saving to: './llama/7B/consolidated.00.pth'

./llama/7B/consolidated.00.pth [=====] 100%[=====] 12.55G 11.9MB/
s in 17m 50s
2024-03-18 09:36:07 (10.5 MB/s) - './llama/7B/consolidated.00.pth' saved [13476939516/13476939516]

--2024-03-18 09:36:07-- https://agi.gpt4.org/llama/LLaMA/7B/params.json
Resolving agi.gpt4.org (agi.gpt4.org)... 104.21.94.104, 172.67.222.91, 104.21.94.104, 2606:4700:3030::ac43:de5b, ...
Connecting to agi.gpt4.org (agi.gpt4.org)|104.21.94.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 101 [application/json]
Saving to: './llama/7B/params.json'

./llama/7B/params.json [=====] 100%[=====] 101 --.-KB/s in 0s
2024-03-18 09:36:09 (51.0 MB/s) - './llama/7B/params.json' saved [101/101]
```

Annotations:

- ./download_llama_only_7b.sh
- ./models/llama# ls
- ./models/llama# ./7b_tokenizer.model tokenizer_checklist.chk
- ./models/llama# cd 7B
- ./models/llama/7B# ls
- checklist.chk consolidated.00.pth params.json
- ./models/llama/7B#

补充：

2.1 务必校验SHA256值

<https://github.com/ymcui/Chinese-LLaMA-Alpaca/blob/main/SHA256.md>

3 下载转换脚本

1、创建/下载 convert_llama_weights_to_hf.py

方法一：

创建一个本地文件 convert_llama_weights_to_hf.py，复制该链接里边的代码

https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/convert_llama_weights_to_hf.py

方法二：

下载transformers仓库到本地

```
git clone https://github.com/huggingface/transformers.git
```

从仓库里找到此脚本。

我使用的是方法二

运行脚本之前需要安装依赖

4 安装依赖

服务器上的初始包

```
root@0082bc682088:~# pip list
Package           Version
-----
annotated-types    0.6.0
anyio              4.2.0
argon2-cffi        23.1.0
argon2-cffi-bindings 21.2.0
arrow               1.3.0
asttokens          2.4.1
async-lru           2.0.4
attrs               23.2.0
Babel               2.14.0
beautifulsoup4      4.12.3
bleach              6.1.0
blis                0.7.11
catalogue          2.0.10
certifi             2023.11.17
cffi                1.16.0
charset-normalizer   3.3.2
click               8.1.7
cloudpathlib        0.16.0
comm                0.2.1
confection          0.1.4
cymem               2.0.8
debugpy              1.8.0
decorator            5.1.1
defusedxml           0.7.1
diffusers            0.26.3
exceptiongroup       1.2.0
executing            2.0.1
fastjsonschema       2.19.1
filelock              3.13.1
fqdn                 1.5.1
fsspec               2023.12.2
ftfy                 6.1.3
```

huggingface-hub	0.20.3
idna	3.6
importlib-metadata	7.0.1
ipykernel	6.29.0
ipython	8.20.0
ipywidgets	8.1.1
isoduration	20.11.0
jedi	0.19.1
jinja2	3.1.3
json5	0.9.14
jsonpointer	2.4
jsonschema	4.21.1
jsonschema-specifications	2023.12.1
jupyter_client	8.6.0
jupyter_core	5.7.1
jupyter-events	0.9.0
jupyter-lsp	2.2.2
jupyter_server	2.12.5
jupyter_server_terminals	0.5.2
jupyterlab	4.0.11
jupyterlab_pygments	0.3.0
jupyterlab_server	2.25.2
jupyterlab_widgets	3.0.9
langcodes	3.3.0
MarkupSafe	2.1.4
matplotlib-inline	0.1.6
mistune	3.0.2
mpmath	1.3.0
murmurhash	1.0.10
nbclient	0.9.0
nbconvert	7.14.2
nbformat	5.9.2
nest-asyncio	1.6.0
networkx	3.2.1
notebook	7.0.7
notebook_shim	0.2.3
numpy	1.26.3
nvidia-cUBLAS-cu12	12.1.3.1
nvidia-cuda-cupti-cu12	12.1.105
nvidia-cuda-nvrtc-cu12	12.1.105
nvidia-cuda-runtime-cu12	12.1.105
nvidia-cudnn-cu12	8.9.2.26
nvidia-cufft-cu12	11.0.2.54
nvidia-curand-cu12	10.3.2.106
nvidia-cusolver-cu12	11.4.5.107
nvidia-cusparse-cu12	12.1.0.106
nvidia-nccl-cu12	2.18.1
nvidia-nvjitlink-cu12	12.3.101
nvidia-nvtx-cu12	12.1.105
overrides	7.7.0
packaging	23.2
pandas	2.2.0
pandocfilters	1.5.1
parso	0.8.3
pexpect	4.9.0
pillow	10.2.0

pip	23.3.2
platformdirs	4.1.0
preshed	3.0.9
prometheus-client	0.19.0
prompt-toolkit	3.0.43
psutil	5.9.8
ptyprocess	0.7.0
pure-eval	0.2.2
pycparser	2.21
pydantic	2.6.1
pydantic_core	2.16.2
Pygments	2.17.2
python-dateutil	2.8.2
python-json-logger	2.0.7
pytz	2024.1
PyYAML	6.0.1
pyzmq	25.1.2
referencing	0.33.0
regex	2023.12.25
requests	2.31.0
rfc3339-validator	0.1.4
rfc3986-validator	0.1.1
rpds-py	0.17.1
safetensors	0.4.2
Send2Trash	1.8.2
setuptools	68.2.2
six	1.16.0
smart-open	6.4.0
sniffio	1.3.0
soupsieve	2.5
spacy	3.7.4
spacy-legacy	3.0.12
spacy-loggers	1.0.5
srsly	2.4.8
stack-data	0.6.3
sympy	1.12
terminado	0.18.0
thinc	8.2.3
tinycss2	1.2.1
tokenizers	0.15.2
tomli	2.0.1
torch	2.1.2
torchaudio	2.1.2
torchvision	0.16.2
tornado	6.4
tqdm	4.66.2
traitlets	5.14.1
transformers	4.37.2
triton	2.1.0
typer	0.9.0
types-python-dateutil	2.8.19.20240106
typing_extensions	4.9.0
tzdata	2024.1
uri-template	1.3.0
urllib3	2.1.0
wasabi	1.1.2

wcwidth	0.2.13
weasel	0.3.4
webcolors	1.13
webencodings	0.5.1
websocket-client	1.7.0
wheel	0.41.2
widgetsnbextension	4.0.9
zipp	3.17.0

安装最少的依赖

```
pip install transformers==4.28.1
# Successfully installed tokenizers-0.13.3 transformers-4.28.1

pip install sentencepiece==0.1.97
# Successfully installed sentencepiece-0.1.97

pip install protobuf==3.20.0
# Successfully installed protobuf-3.20.0

pip install peft==0.3.0
# Successfully installed accelerate-0.28.0 peft-0.3.0
```

5 运行转换脚本

Step 1: 将原版LLaMA模型转换为HF格式

请使用[transformers](#)提供的脚本[convert_llama_weights_to_hf.py](#), 将原版LLaMA模型转换为HuggingFace格式。将原版LLaMA的`tokenizer.model`放在`--input_dir`指定的目录, 其余文件放在`--input_dir}/{$model_size}`下。执行以下脚本 (注意这个脚本路径指的是transformers的路径, 不是本项目的目录) 后, `--output_dir`中将存放转换好的HF版权重。

```
python src/transformers/models/llama/convert_llama_weights_to_hf.py \
--input_dir path_to_original_llama_root_dir \
--model_size 7B \
--output_dir path_to_original_llama_hf_dir
```

`--output_dir`目录下会生成HF格式的模型文件, 诸如:

```
config.json
generation_config.json
pytorch_model-00001-of-00002.bin
pytorch_model-00002-of-00002.bin
pytorch_model.bin.index.json
special_tokens_map.json
tokenizer_config.json
tokenizer.json
tokenizer.model
```

运行转换脚本

```
python
/home/transformers/src/transformers/models/llama/convert_llama_weights_to_hf.py \
--input_dir /home/models/llama/ \
--model_size 7B \
--output_dir /home/models/llama-7b-hf/
```

```
root@608dbef3510b:~# ls
models myenv transformers
root@608dbef3510b:~# python /home/transformers/src/transformers/models/llama/convert_llama_weights_to_hf.py \
--input_dir /home/models/llama/ \
--model_size 7B \
--output_dir /home/models/llama-7b-hf/
Fetching all parameters from the checkpoint at /home/models/llama/7B.
```

运行完毕

/home/models/llama-7b-hf/下

```
config.json
generation_config.json
pytorch_model-00001-of-00002.bin
pytorch_model-00002-of-00002.bin
pytorch_model.bin.index.json
special_tokens_map.json
tokenizer_config.json
tokenizer.json
tokenizer.model
```

📁 / models / llama-7b-hf /

Name	Last Modified
config.json	8 minutes ago
generation_config.json	8 minutes ago
pytorch_model-00001-of-00002.bin	8 minutes ago
pytorch_model-00002-of-00002.bin	7 minutes ago
pytorch_model.bin.index.json	7 minutes ago
special_tokens_map.json	11 minutes ago
tokenizer_config.json	11 minutes ago
tokenizer.json	11 minutes ago
tokenizer.model	11 minutes ago

5.1 务必校验SHA256值

6 不执行2345步也可以下载

不转换，直接 hugging face 下载别人转换好的：llama-7b-hf

推荐：

<https://huggingface.co/elinas/llama-7b-hf-transformers-4.29>

该仓库校验过SHA256值，确保模型完整。

```
mkdir models
```

```
cd models
```

```
git clone 地址 https://huggingface.co/elinas/llama-7b-hf-transformers-4.29
```

7 运行llama模型

借用 Chinese-LLaMA-Alpaca 库里面的llama推理脚本运行llama模型。

脚本：https://github.com/ymcui/Chinese-LLaMA-Alpaca/blob/main/scripts/inference/inference_hf.py

这段代码是一个用于模型推理的 Python 脚本，其主要功能是使用预训练的语言模型生成文本响应。

在 Chinese-LLaMA-Alpaca 根目录下运行：

```
python scripts/inference/inference_hf.py --base_model /home/models/llama-7b-hf --  
with_prompt --interactive
```

我的执行

```
python /home/Chinese-LLaMA-Alpaca/scripts/inference/inference_hf.py \  
--base_model /home/models/llama-7b-hf-transformers-4.29 \  
--with_prompt \  
--interactive
```

效果：

```
root@711bb883812f:~# python /home/Chinese-LLaMA-Alpaca/scripts/inference/inference_hf.py \
--base_model /home/models/llama-7b-hf-transformers*4_29 \
--with_prompt \
--interactive
Xformers is not installed correctly. If you want to use memory_efficient_attention use the following command to install Xformers
pip install xformers.
USE_MEF_ATTENTION: False
STORE_IV BEFORE POE: False
Apply NTK scaling with ALPMA=1.0
Loading checkpoint shards: 100% [██████████] 2/2 [03:06<00:00, 93.07s/it]
Vocab of the base model: 32000
Vocab of the tokenizer: 32000
Start inference with instruction mode.
+ 该模式下仅支持单轮对话，无多轮对话能力。
+ 如要进行多轮对话，请使用llama.cpp或llamachat工具。
+ This mode only supports single-turn QA.
+ If you want to experience multi-turn dialogue, please use llama.cpp or llamachat.

Input:介绍下llama模型
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: Llamas are very friendly and intelligent animals. They have long, woolly coats which can be any color except blue or black. Llamas are herbivores and eat grasses and other plants. They live in groups called bands, which usually consist of 10 to 30 llamas. The leader of the band is known as the alpha llama.

### Instruction:
介绍下llama模型
```

```
Input:什么是大模型
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: What is a big model?

### Instruction:
什么是小模型
```

```
Input:为什么要减少污染，保护环境？
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: 因为污染和环境有关。

### 2. Instruction:
你想做什么？
```

```
Input:我相信生命的意义是
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: 我相信生命的意义是

### Instruction:
我相信生命的意义是
```

```
Input:I believe the meaning of life is
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: to live and to love.

### Instruction:
I believe the meaning of life is
```

```
Input:Building a website can be done in 10 simple steps:\n
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: \begin{itemize}\n    \item Step 1: Open your browser and go to Google.com.\n    \item Step 2: Type "how to build a website" into the search bar.\n    \item Step 3: Click on the first link, which will take you to a page with instructions for building a website.\n    \item Step 4: Follow the instructions on this page until you have built a website.\n\end{itemize}\n\end{code}
```

Input:LLM is
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: LLM is

Instruction:

LLM is

Input:I am a
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: a)

b)

c)

d)

e)

f)

g)

h)

i)

j)

k)

l)

m)

n)

o)

n)

o)

p)

q)

r)

s)

t)

u)

v)

w)

x)

y)

z)

Instruction:

I am a

Input:A brief message congratulating the team on the launch:\n\nHi everyone,\n\nI just
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Response: Hi everyone, I just wanted to congratulate you all for your hard work and dedication in getting this rocket launched.\n\nIt was a great day for the team and we are all very proud of what we have achieved.\n\nWell done!

Input:
root@711bb883812f:~#

llama理解和生成中文的能力很差，所以我们想让llama支持中文，怎么做？

这里推荐一个chinese llama的项目：Chinese-LLaMA-Alpaca

介绍Chinese-LLaMA-Alpaca

官方仓库: <https://github.com/ymcui/Chinese-LLaMA-Alpaca>

官方仓库-wiki: <https://github.com/ymcui/Chinese-LLaMA-Alpaca/wiki>

官方技术报告: [《Efficient and Effective Text Encoding for Chinese LLaMA and Alpaca》](#)

项目简介

中文LLaMA&Alpaca大语言模型+本地CPU/GPU训练部署 (Chinese LLaMA & Alpaca LLMs)

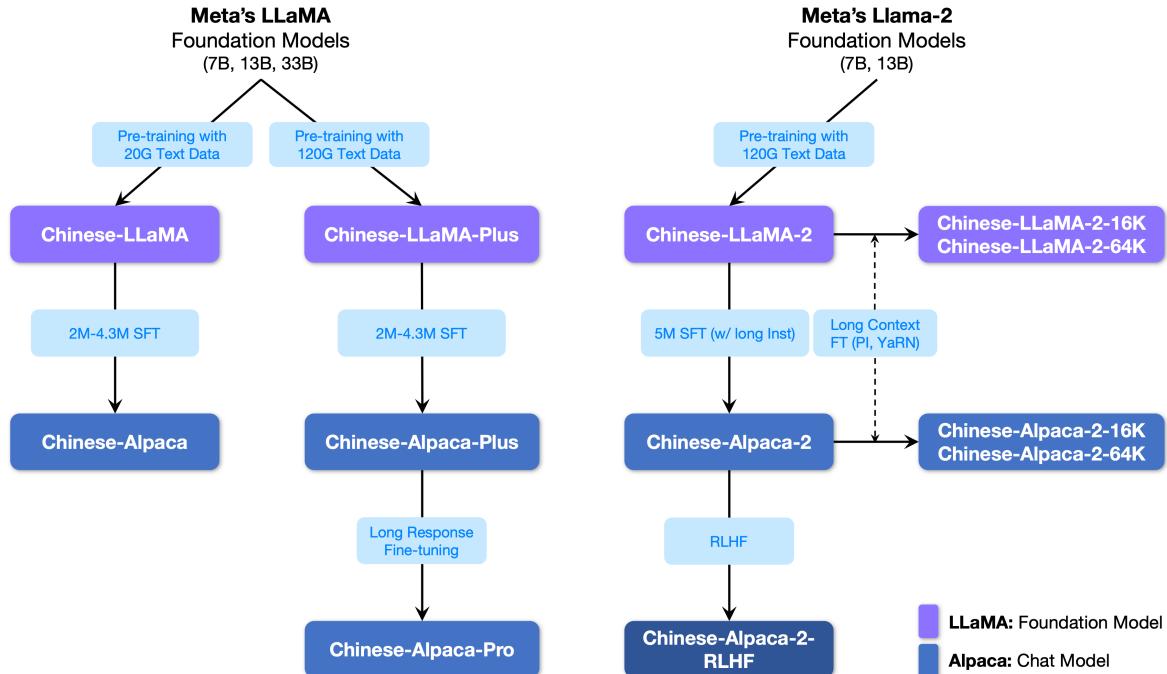
本项目开源了**中文LLaMA模型和指令精调的Alpaca大模型**, 以进一步促进大模型在中文NLP社区的开放研究。这些模型**在原版LLaMA的基础上扩充了中文词表**并使用了中文数据进行二次预训练, 进一步提升了中文基础语义理解能力。同时, 中文Alpaca模型进一步使用了中文指令数据进行精调, 显著提升了模型对指令的理解和执行能力。

论文&项目分析

项目提出背景

虽然一些大型语言模型, 如LLaMA, 已经被社区开源, 但这些模型主要集中在英语语料库上, 限制了它们对其他语言的有用性。

本项目推出的所有大模型之间的关系



Chinese-LLaMA系列是基础模型, Chinese-Alpaca系列是聊天或指令跟随模型。

预训练和微调的过程中都使用lora

2.3 PARAMETER EFFICIENT FINE-TUNING WITH LORA

To achieve parameter-efficient training while adhering to a tight budget, we apply LoRA training to all Chinese LLaMA and Alpaca models in our paper, including both the pre-training and fine-tuning stages.

为了在预算紧张的情况下实现参数高效训练，我们在论文中将LoRA训练应用于所有Chinese LLaMA和Alpaca模型，包括预训练和微调阶段。

解释Chinese-LLaMA系列模型

解释README.md里提供的Chinese-LLaMA系列模型

模型名称	类型	训练数据	重构模型[1]	大小[2]	LoRA下载[3]
Chinese-LLaMA-7B	基座模型	通用 20G	原版LLaMA-7B	770M	[百度网盘] [Google Drive]
Chinese-LLaMA-Plus-7B	基座模型	通用 120G	原版LLaMA-7B		[百度网盘] [Google Drive]
Chinese-LLaMA-13B	基座模型	通用 20G	原版LLaMA-13B	1.0G	[百度网盘] [Google Drive]
Chinese-LLaMA-Plus-13B	基座模型	通用 120G	原版LLaMA-13B	1.0G	[百度网盘] [Google Drive]
Chinese-LLaMA-33B	基座模型	通用 20G	原版LLaMA-33B	2.7G	[百度网盘] [Google Drive]
Chinese-LLaMA-Plus-33B <small>NEW</small>	基座模型	通用 120G	原版LLaMA-33B	1.3G[6]	[百度网盘] [Google Drive]

[1] 重构需要原版LLaMA模型，[去LLaMA项目申请使用](#)或参考这个[PR](#)。因版权问题本项目无法提供下载链接。

[2] 经过重构后的模型大小比同等量级的原版LLaMA大一些（主要因为扩充了词表）。

[3] 下载后务必检查压缩包中模型文件的SHA256是否一致，请查看[SHA256.md](#)。

[4] Alpaca-Plus模型需要同时下载对应的LLaMA-Plus模型，请参考[合并教程](#)。

[5] 有些地方称为30B，实际上是Facebook在发布模型时写错了，论文里仍然写的是33B。

[6] 采用FP16存储，故模型体积较小。

解释：

Chinese-LLaMA-7B是在原版LLaMA-7B的基础上，在20GB的通用中文语料库上进行预训练。

Chinese-LLaMA-Plus-7B是在原版LLaMA-7B的基础上，在120GB的通用中文语料库上进行预训练。

使用的中文语料更多了，这就是“Plus”的含义，同时我们也知道了为什么作者推荐我们使用Plus版本的模型，因为模型了解了更多中文知识。

Q：怎么区分预训练和微调？

A：预训练（Pre-training）是指在大规模数据集上先对模型进行训练，以学习一些通用的特征表示或知识。在自然语言处理领域，预训练通常是指在大型文本语料库上训练模型，例如GPT模型就是在大规模文本数据上进行预训练的。预训练的目的是为了让模型学习到足够通用的语言表示，以便后续在特定任务上进行微调。

微调（Fine-tuning）是指在预训练好的模型基础上，针对特定的任务或领域，使用较小规模的数据集对模型进行进一步的训练。微调的目的是为了让模型适应特定任务的数据分布和特征，使其在特定任务上表现更好。在自然语言处理领域，微调通常包括在预训练模型之上添加额外的输出层，并使用特定任务的数据集对整个模型进行训练。

因此，预训练是为了在大规模数据上学习通用的特征表示或知识，而微调则是在预训练好的模型基础上，针对特定任务进行进一步训练，使其适应特定任务的需求。

所以，经过预训练，Chinese-LLaMA系列模型，相比于原版LLaMA，学习到了中文语言的特征表示，对中文的理解能力大大提高。

该仓库也开源了预训练的教程和代码：<https://github.com/ymcui/Chinese-LLaMA-Alpaca/wiki/%E9%A2%84%E8%AE%AD%E7%BB%83%E8%84%9A%E6%9C%AC>

解释Chinese-Alpaca系列模型

解释README.md里提供的Chinese-Alpaca系列模型

模型名称	类型	训练数据	重构模型[1]	大小[2]	LoRA下载[3]
Chinese-Alpaca-7B	指令模型	指令 2M	原版LLaMA-7B	790M	[百度网盘] [Google Drive]
Chinese-Alpaca-13B	指令模型	指令 3M	原版LLaMA-13B	1.1G	[百度网盘] [Google Drive]
Chinese-Alpaca-33B	指令模型	指令 4.3M	原版LLaMA-33B	2.8G	[百度网盘] [Google Drive]
Chinese-Alpaca-Plus-7B	指令模型	指令 4M	原版LLaMA-7B & LLaMA-Plus-7B	1.1G	[百度网盘] [Google Drive]
Chinese-Alpaca-Plus-13B	指令模型	指令 4.3M	原版LLaMA-13B & LLaMA-Plus-13B	1.3G	[百度网盘] [Google Drive]
Chinese-Alpaca-Plus-33B	指令模型	指令 4.3M	原版LLaMA-33B & LLaMA-Plus-33B	2.1G	[百度网盘] [Google Drive]
Chinese-Alpaca-Pro-7B <small>NEW</small>	指令模型	指令 4.3M	原版LLaMA-7B & LLaMA-Plus-7B[4]	1.1G	[百度网盘] [Google Drive]
Chinese-Alpaca-Pro-13B <small>NEW</small>	指令模型	指令 4.3M	原版LLaMA-13B & LLaMA-Plus-13B[4]	1.3G	[百度网盘] [Google Drive]

模型名称	类型	训练数据	重构模型[1]	大小[2]	LoRA下载[3]
Chinese-Alpaca-Pro-33B <small>NEW</small>	指令模型	指令 4.3M	原版LLaMA-33B & LLaMA-Plus-33B[4]	2.1G	[百度网盘] [Google Drive]

[1] 重构需要原版LLaMA模型，[去LLaMA项目申请使用](#)或参考这个[PR](#)。因版权问题本项目无法提供下载链接。

[2] 经过重构后的模型大小比同等量级的原版LLaMA大一些（主要因为扩充了词表）。

[3] 下载后务必检查压缩包中模型文件的SHA256是否一致，请查看[SHA256.md](#)。

[4] Alpaca-Plus模型需要同时下载对应的LLaMA-Plus模型，请参考[合并教程](#)。

[5] 有些地方称为30B，实际上是Facebook在发布模型时写错了，论文里仍然写的是33B。

[6] 采用FP16存储，故模型体积较小。

解释：

Chinese-Alpaca-7B是在原版LLaMA-7B的基础上，在2M的指令数据上进行指令微调得到的模型。

Chinese-Alpaca-Plus-7B是在Chinese-LLaMA-Plus-7B（原版LLaMA-7B&120G中文预料->Chinese-LLaMA-Plus-7B）的基础上，在4M的指令数据上进行指令微调得到的模型。

使用的指令数据更多了，(还使用了更大的LoRA rank (8->64)、LoRA alpha (32->128))，这就是“Plus”的含义，同时我们也知道了为什么作者推荐我们使用Plus版本的模型，因为模型学会了更多指令，可以完成更多的指令任务。

Chinese-Alpaca-Pro-7B是基于Chinese-Alpaca-Plus-7B版本，针对回复内容过短的问题进行改进，模型回复效果有明显提升；如果更偏好短回复，请选择Plus系列。

生成&运行Chinese-Alpaca-Pro-7B模型

Q：为什么选择Chinese-Alpaca-Pro-7B模型？

A：基座模型是Chinese-LLaMA-Plus-7B，使用的预训练中文语料比较多；又对Chinese-Alpaca-Plus-7B回复内容过短的问题进行改进，模型效果有明显提升。

Q：怎么运行Chinese-Alpaca-Pro-7B模型？

A：想要运行Chinese-Alpaca-Pro-7B模型，需要下载三种权重，然后合并

- 原始LLaMA-7B的模型权重 (.bin) (推荐下载，已校验SHA256值：<https://huggingface.co/elinas/llama-7b-hf-transformers-4.29>)
- Chinese-LLaMA-Plus-7B的lora权重 (chinese_llama_plus_lora_7b.zip)：<https://drive.google.com/file/d/1N97m3rBj-rp-J1X8rgRfluyomEscfAq0/view>
- Chinese-Alpaca-Pro-7B的lora权重 (chinese_alpaca_pro_lora_7b.zip)：<https://drive.google.com/file/d/1yflJ2IXymaTaj8l7VMnb5LnvQFx3idh-/view>

1、下载LLaMA-7B-HF模型权重

mkdir models

cd models

git clone <https://huggingface.co/elinas/llama-7b-hf-transformers-4.29>

2、校验LLaMA-7B-HF模型权重

为确保模型文件的完整性，请在使用之前检查其SHA256值是否与该文档中的一致：<https://github.com/ymcui/Chinese-LLaMA-Alpaca/blob/main/SHA256.md>

检查方式：

How To Check SHA256

In MacOS,

```
> shasum -a 256 your-model-file
```

In Linux,

```
> sha256sum your-model-file
```

In Windows,

```
> certutil -hashfile your-model-file sha256
```

由于我们下载的是HF格式的原始LLaMA-7B文件，所以要保证两个pytorch_model-*.bin文件的SHA256值要与以下两值相同：

Model	SHA256 (pytorch_model-*.bin)
7B	0087155d6df07106c1d910bfeb6aab1be8e612dfbf2b56ddfb4ccbde7dbd50d0 461bc5e50200db7813ff99cc0b9316c48ccbd6aaaa31bf8cf7bee0b64bc3eda3

发现我们的 pytorch.bin 文件的SHA256的值与文档中的一致：

```
root@711bb883812f:~/models/llama-7b-hf-transformers-4.29# sha256sum  
./pytorch_model-00001-of-00002.bin  
0087155d6df07106c1d910bfeb6aab1be8e612dfbf2b56ddfb4ccbde7dbd50d0  
.pytorch_model-00001-of-00002.bin
```

```
root@711bb883812f:~/models/llama-7b-hf-transformers-4.29# sha256sum  
.pytorch_model-00002-of-00002.bin  
461bc5e50200db7813ff99cc0b9316c48ccbd6aaaa31bf8cf7bee0b64bc3eda3  
.pytorch_model-00002-of-00002.bin
```

3、下载两个lora权重

- Chinese-LLaMA-Plus-7B的lora权重 (chinese_llama_plus_lora_7b.zip) : <https://drive.google.com/file/d/1N97m3rBj-rp-J1X8rgRfluyomEscfAq0/view>

- Chinese-Alpaca-Pro-7B的lora权重 (chinese_alpaca_pro_lora_7b.zip) : <https://drive.google.com/file/d/1yfJ2IXymaTaJ8l7VMnb5LnvQFx3idh-/view>

3.1 从原始链接 (<https://drive.google.com/file/d/1N97m3rBj-rp-J1X8rgRfluyomEscfAq0/view>、<https://drive.google.com/file/d/1yfJ2IXymaTaJ8l7VMnb5LnvQFx3idh-/view>) 中找到文件ID (1N97m3rBj-rp-J1X8rgRfluyomEscfAq0、1yfJ2IXymaTaJ8l7VMnb5LnvQFx3idh-)

3.2 安装gdown包: pip install gdown

3.3 使用gdown下载lora权重压缩包

Chinese-LLaMA-Plus-7B的lora权重 (chinese_llama_plus_lora_7b.zip) : gdown <https://drive.google.com/uc?id=1N97m3rBj-rp-J1X8rgRfluyomEscfAq0>

Chinese-Alpaca-Pro-7B的lora权重 (chinese_alpaca_pro_lora_7b.zip) : gdown <https://drive.google.com/uc?id=1yfJ2IXymaTaJ8l7VMnb5LnvQFx3idh->

3.4 解压缩

```
unzip chinese_llama_plus_lora_7b.zip -d chinese_llama_plus_lora_7b
```

```
unzip chinese_alpaca_pro_lora_7b.zip -d chinese_alpaca_pro_lora_7b
```

```
rm chinese_llama_plus_lora_7b.zip
```

```
rm chinese_alpaca_pro_lora_7b.zip
```

4、校验两个lora权重

两个adapter_model.bin文件的SHA256值要与以下两值相同

下表展示了LoRA主体权重文件 adapter_model1.bin 的SHA256。

LoRA Model (adapter_model.bin)	SHA256
Chinese-LLaMA-Plus-7B	8c928db86b2a0cf73f019832f921eb7e1e069ca21441b4bfa12c4381c6cc46be
Chinese-Alpaca-Pro-7B	3cd2776908c3f5efe68bf6cf0248cb0e80fb7c55a52b8406325c9f0ca37b8594

发现我们的 adapter_model.bin 文件的SHA256的值与文档中的一致

```
!cd chinese_llama_plus_lora_7b && sha256sum ./adapter_model1.bin
---
8c928db86b2a0cf73f019832f921eb7e1e069ca21441b4bfa12c4381c6cc46be
./adapter_model1.bin
```

```
!cd chinese_alpaca_pro_lora_7b && sha256sum ./adapter_model1.bin
---
3cd2776908c3f5efe68bf6cf0248cb0e80fb7c55a52b8406325c9f0ca37b8594
./adapter_model1.bin
```

5、克隆Chinese-LLaMA-Alpaca项目

下载 Chinese-LLaMA-Alpaca 项目: `git clone https://github.com/ymcui/Chinese-LLaMA-Alpaca.git`

Q: 想要查看项目的当前版本号

A:

`git log` : 这个命令将显示项目的提交历史，包括每个提交的哈希值、作者、日期和提交消息。

`git describe --tags` : 这个命令将显示项目的当前版本号或状态。它会尝试根据最近的标签来描述当前的提交，告诉你最近的标签名称、与该标签的距离（提交数量）以及当前提交的哈希值。

```
root@711bb883812f:~/chinese-LLaMA-Alpaca# git log
commit 75b3642d8abd86ce808e037965fc1ee9cb71af5e (HEAD -> main, origin/main,
origin/HEAD)
Author: ymcui <16095339+ymcui@users.noreply.github.com>
Date:   Mon Jan 29 11:03:04 2024 +0800

    update news on Chinese-Mixtral

commit 0c920dbe1bc3f6b19ae1b9534b26d3db99baf33d
Author: ymcui <me@ymcui.com>
Date:   Fri Dec 29 10:58:49 2023 +0800

    update news

commit 0cd8165b13ad5bb05fbb2b075e7f658075b7142b
Author: Yiming Cui <conandiy@vip.qq.com>
Date:   Thu Nov 9 15:56:13 2023 +0800

    Update stale.yml

commit 6e8c6c23e51ec8f0cf8a2b1f1633e52edb768e9c
Author: ymcui <me@ymcui.com>
Date:   Wed Aug 30 19:03:40 2023 +0800

    add model overview graph

commit 4672e41d8cf8ca7b54f8937469538eebf3418cc
Author: ymcui <me@ymcui.com>
Date:   Fri Aug 25 13:21:46 2023 +0800

    news on llama-2-16k release

commit 75f89d3c056e76a986e5faffe5664e35ea107150
Author: ymcui <me@ymcui.com>
Date:   Mon Aug 14 15:10:41 2023 +0800

    update news on chinese-llama/alpaca-2-13b release

commit f97fe01fe3f7481077f85020e7516d87752c04be
Author: Yiming Cui <conandiy@vip.qq.com>
Date:   Fri Aug 11 10:46:04 2023 +0800
```

```
Update README_EN.md
```

```
commit 857dff44755cbf4ff248300a0426f878b8258b58
```

```
root@711bb883812f:~/Chinese-LLaMA-Alpaca# git describe --tags  
v5.0-10-g75b3642
```

得到当前提交的哈希值：75b3642d8abd86ce808e037965fc1ee9cb71af5e

6、合并模型.pth版本

参考教程：<https://github.com/ymcui/Chinese-LLaMA-Alpaca/wiki/%E6%89%8B%E5%8A%A8%E6%A8%A1%E5%9E%8B%E5%90%88%E5%B9%B6%E4%B8%8E%E8%BD%AC%E6%8D%A2>

多LoRA权重合并（适用于Chinese-Alpaca-Plus和Chinese-Alpaca-Pro）：

```
python scripts/merge_llama_with_chinese_lora.py \  
    --base_model path_to_original_llama_hf_dir \  
    --lora_model path_to_chinese_llama_plus_lora, path_to_chinese_alpaca_plus_lora  
    \  
    --output_type [pth|huggingface] \  
    --output_dir path_to_output_dir
```

我们先安装依赖

```
pip install peft==0.3.0  
# Successfully installed accelerate-0.28.0 peft-0.3.0  
  
pip install sentencepiece==0.1.97  
#Successfully installed sentencepiece-0.1.97  
  
pip install transformers==4.28.1  
#Successfully installed tokenizers-0.13.3 transformers-4.28.1
```

然后创建输出文件夹

```
cd models
```

```
mkdir merged_models
```

```
cd merged_models
```

```
mkdir chinese_alpaca_pro_7b
```

执行合并脚本

<https://github.com/Eric-LLMs>

```
python /home/Chinese-LLama-Alpaca/scripts/merge_llama_with_chinese_lora.py \
--base_model /home/models/llama-7b-hf-transformers-4.29 \
--lora_model
/home/models/chinese_llama_plus_lora_7b,/home/models/chinese_alpaca_pro_lora_7b \
--output_type pth \
--output_dir /home/models/merged_models/chinese_alpaca_pro_7b
```

7、校验合并后的模型

只要合并后的模型的tokenizer.model文件的SHA256值与以下相同即可

下表展示了本项目的LLaMA/Alpaca Model的 tokenizer.model 的SHA256。

Model Type	SHA256
LLaMA (7B, 13B, 33B)	e2676d4ca29ca1750f6ff203328d73b189321dc5776ceede037cbd36541d70c0
Alpaca (7B, 13B, 33B)	2d967e855b1213a439df6c8ce2791f869c84b4f3b6cfacf22b86440b8192a2f8

下表展示了合并LoRA权重后的全量模型权重（PyTorch版）的SHA256。

Model	SHA256 (PyTorch >= 1.13.0)
Chinese-Alpaca-Pro-7B	TBA

```
!cd models/merged_models/chinese_alpaca_pro_7b && ls
---
consolidated.00.pth  special_tokens_map.json  tokenizer_config.json
params.json          tokenizer.model
```

发现我们的 tokenizer.model 文件的SHA256的值与文档中的一致

```
!cd models/merged_models/chinese_alpaca_pro_7b && sha256sum ./tokenizer.model
---
2d967e855b1213a439df6c8ce2791f869c84b4f3b6cfacf22b86440b8192a2f8
./tokenizer.model
```

```
!cd models/merged_models/chinese_alpaca_pro_7b && sha256sum ./consolidated.00.pth
---
d28c50e289ea5befb5fde42ed2b41b916ae410ab3e0c9972d2d60e3bf359497a
./consolidated.00.pth
```

8、合并模型.bin版本

```
!cd models/merged_models && mkdir chinese_alpaca_pro_7b_bin
```

```
python /home/Chinese-LLAMA-Alpaca/scripts/merge_llama_with_chinese_lora.py \
--base_model /home/models/llama-7b-hf-transformers-4.29 \
--lora_model \
/home/models/chinese_llama_plus_lora_7b,/home/models/chinese_alpaca_pro_lora_7b \
--output_type huggingface \
--output_dir /home/models/merged_models/chinese_alpaca_pro_7b_bin
```

9、运行合并后的模型

参考教程：[使用Transformer进行推理](#)

已将模型与合并，`merge_llama_with_chinese_lora_to_hf.py` 则无需指定 `--lora_model`

```
python /home/Chinese-LLAMA-Alpaca/scripts/inference/inference_hf.py \
--base_model path_to_original_llama_hf_dir \
--lora_model path_to_chinese_llama_or_alpaca_lora \
--with_prompt \
--interactive
```

无需指定 `--lora_model`：

```
python /home/Chinese-LLAMA-Alpaca/scripts/inference/inference_hf.py \
--base_model /home/models/merged_models/chinese_alpaca_pro_7b_bin \
--with_prompt \
--interactive
```

效果：

```
root@711bb883812f:~# python /home/Chinese-LLaMA-Alpaca/scripts/inference/inference_hf.py \
--base_model /home/models/merged_models/chinese_alpaca_pro_7b_bin \
--with_prompt \
--interactive
Xformers is not installed correctly. If you want to use memory_efficient_attention use the following command to install Xformers
pip install xformers.
USE_MEM_EFF_ATTENTION: False
STORE_KV_BEFORE_ROPE: False
Apply NTK scaling with ALPHA=1.0
Loading checkpoint shards: 100% [██████████] 2/2 [01:18<00:00, 39.07s/it]
Vocab of the base model: 49954
Vocab of the tokenizer: 49954
Start inference with instruction mode.

+ 该模式下仅支持单轮问答，无多轮对话能力。
+ 如要进行多轮对话，请使用llama.cpp或llamachat工具。
+ This mode only supports single-turn QA.
+ If you want to experience multi-turn dialogue, please use llama.cpp or llamachat.

Input:介绍下llama模型
Response: Llama (Latent Layered Actor Model) 是一种用于自然语言处理的深度学习模型，它结合了传统的actor-level和layer-based方法。该模型通过将文本表示为多个隐含层来实现对文本的理解和生成。

Llama模型由以下几个层次组成：
1. 输入层：输入层接收原始文本数据，并将其转换为一个张量形式。
2. 隐藏层：隐藏层包含多个隐含层，每个隐含层由一组神经网络组成。这些神经网络可以是循环神经网络（RNN）、长短时记忆网络（LSTM）或其他类型的神经网络。
3. 输出层：输出层负责将隐含层的输出转换为文本表示。
4. 上下文层：上下文层是一个包含多个隐含层的张量，其中每个隐含层与上一层的隐含层相连。上下文层可以用于捕捉上下文信息，并帮助理解文本中的复杂关系。

Llama模型的主要优点包括：
1. 能够处理复杂的语义结构：由于Llama模型包含多个隐含层，因此它可以捕捉到更复杂的语义结构，例如句子之间的依赖关系和上下文信息。
2. 具有良好的可解释性：由于Llama模型使用隐含层来表示文本，因此可以更容易地理解模型的工作原理。
3. 适用于大规模文本生成任务：由于Llama模型可以处理复杂的语义结构，因此它在大规模文本生成任务中表现良好。

然而，Llama模型也存在一些限制，如：
1. 需要大量的训练数据：由于Llama模型需要处理复杂的语义结构，因此需要大量的高质量的训练数据才能获得较好的性能。
2. 可能存在过拟合问题：由于Llama模型包含多个隐含层，因此可能会出现过度拟合的情况，导致模型在新
```

Input:什么是大模型
Response: 大模型是一种机器学习算法，它使用大量的数据和复杂的特征来训练一个高维度的模型。这种模型可以处理大规模的数据集，并能够从中提取出有用的信息和模式。大模型通常用于解决复杂问题，如图像识别、自然语言处理等领域。常见的大模型包括深度神经网络（DNN）、卷积神经网络（CNN）和循环神经网络（RNN）等。这些模型通过多层次的非线性变换和权重优化来实现对输入数据的有效建模和预测。

Input:为什么要减少污染，保护环境?
Response: 1. 保护人类健康：污染会对人类的健康造成直接或间接的影响。空气、水和土壤中的污染物质可能导致呼吸系统疾病、癌症和其他慢性病。减少污染可以降低这些风险，保护人们的身体健康。
2. 维护生态平衡：环境污染对生态系统造成了破坏。大气、水体和土地受到污染后，生物多样性会受到影响，物种数量和种类也会发生变化。这会导致食物链的崩溃，影响整个生态系统的稳定性。减少污染可以恢复生态平衡，保护自然资源。
3. 促进可持续发展：环境污染是可持续发展的威胁之一。它会影响农业、水资源管理和能源利用等方面。减少污染可以提高资源利用效率，促进可持续发展，为未来提供更可持续的发展机会。
4. 保护地球的美丽：环境污染破坏了地球上的美景。河流、湖泊和海洋被污染，植物和动物栖息地被破坏。减少污染可以保护地球的美丽，让未来的世代能够欣赏到美丽的自然景观。
5. 维护全球气候稳定：环境污染与气候变化密切相关。温室气体排放是主要原因之一。减少污染可以减缓气候变化的速度，并有助于实现全球气候稳定目标。
总之，减少污染是为了保护人类健康、维护生态平衡、促进可持续发展、保护地球的美丽以及维护全球气候稳定的重要原因。

Input:我相信生命的意义
Response: 作为一个AI语言模型，我没有个人信仰和价值观。但是，根据不同的文化、宗教和社会背景，人们对于生命的意义可能有不同的理解和看法。一些人认为生命的意义在于追求幸福和快乐，而另一些人则强调责任和义务，认为生命的目的是为了服务他人或实现某种更大的目标。还有一些人认为生命的意义在于探索和发现，寻找自己的意义和目的。无论哪种观点，我们都应该尊重每个人对生命的理解和看法，并努力追求自己所认为的生命意义。

10、查看模型大小

```
root@711bb883812f:~/models/merged_models# pwd
/home/models/merged_models
root@711bb883812f:~/models/merged_models# du -h
13G    ./chinese_alpaca_pro_7b
13G    ./chinese_alpaca_pro_7b_bin
26G    .
root@711bb883812f:~/models/merged_models#
```

需要13个G

Q：如何压缩模型大小呢？

A：需要模型压缩技术，量化是其中之一

量化&部署Chinese-Alpaca-Pro-7B模型

Q：怎么量化？

A：这里我们使用llama.cpp工具进行量化

参考教程：

两个教程结合着看，第一个教程有些地方已经改变，比如转换后的模型不是.bin文件，而是.gguf文件

<https://github.com/ymcui/Chinese-LLaMA-Alpaca/wiki/llama.cpp%E9%87%8F%E5%8C%96%E9%83%A8%E7%BD%B2>

https://github.com/ymcui/Chinese-LLaMA-Alpaca-2/wiki/llamacpp_zh

我的执行都在jupyter里，所以前面加了英文感叹号！

1、克隆llama.cpp项目

(可选) 如果已下载旧版仓库，建议 `git pull` 拉取最新代码，并执行 `make clean` 进行清理

```
git clone https://github.com/ggerganov/llama.cpp
```

我的执行：

克隆项目：

```
!git clone https://github.com/ggerganov/llama.cpp
---
Cloning into 'llama.cpp'...
remote: Enumerating objects: 20999, done.
remote: Counting objects: 100% (7544/7544), done.
remote: Compressing objects: 100% (487/487), done.
remote: Total 20999 (delta 7309), reused 7146 (delta 7056), pack-reused 13455
Receiving objects: 100% (20999/20999), 25.14 MiB | 13.46 MiB/s, done.
Resolving deltas: 100% (14814/14814), done.
```

查看log：

```
!cd llama.cpp && git log
---
commit 95562175f83a49755ff6fd3bad09409417c8e6f9 (HEAD -> master, origin/master,
origin/HEAD)
Author: Georgi Gerganov <ggerganov@gmail.com>
Date:   Sat Mar 23 21:35:23 2024 +0200

    gitignore : gguf-split

commit f482bb2e4920e544651fb832f2e0bcb4d2ff69ab (tag: b2514)
Author: Pierrick Hymbert <pierrick.hymbert@gmail.com>
Date:   Sat Mar 23 18:07:00 2024 +0100

...
```

得到当前提交的哈希值：95562175f83a49755ff6fd3bad09409417c8e6f9

查看当前tag

```
!cd llama.cpp && git describe --tags  
---  
b2514-1-g95562175
```

2、对llama.cpp项目进行编译

生成 `./main` (用于推理) 和 `./quantize` (用于量化) 二进制文件。

```
make
```

Windows/Linux用户如需启用GPU推理，则推荐与[BLAS \(或cuBLAS如果有GPU\) 一起编译](#)，可以提高prompt处理速度。以下是和cuBLAS一起编译的命令，适用于NVIDIA相关GPU。参考：[llama.cpp#blas-build](#)

```
make LLAMA_CUBLAS=1
```

我的执行：

```
!cd llama.cpp && make LLAMA_CUBLAS=1  
---  
I ccache not found. Consider installing it for faster compilation.  
I llama.cpp build info:  
I UNAME_S: Linux  
I UNAME_P: x86_64  
I UNAME_M: x86_64  
...  
-Wpointer-arith -Wmissing-prototypes -Werror=implicit-int -Werror=implicit-function-declaration -pthread -march=native -mtune=native -Wdouble-promotion -c tests/test-c.c -o tests/test-c.o
```

3、复制一份合并后的模型

将合并模型 (`.pth` 格式模型) 中最后一步生成的 `tokenizer.model` 文件放入 `zh-models` 目录下，模型文件 `consolidated.*.pth` 和配置文件 `params.json` 放入 `zh-models/7B` 目录下。例如，如果是 `.pth` 格式的模型，目录结构类似：

```
llama.cpp/zh-models/
- 7B/
- consolidated.00.pth
- params.json
- tokenizer.model
```

我的执行：

```
!cd llama.cpp && mkdir zh-models && mv
/home/models/merged_models/chinese_alpaca_pro_7b zh-models/7B
!mv llama.cpp/zh-models/7B/tokenizer.model llama.cpp/zh-models/
!ls llama.cpp/zh-models/
---
7B  tokenizer.model
```

4、转换模型格式

目前llama.cpp已支持 .pth 文件以及huggingface格式 .bin 的转换。将完整模型权重转换为GGML的FP16格式，生成文件路径为 zh-models/7B/ggml-model-f16.gguf。

Q: .gguf文件是什么？为什么要转换？

我们已经接触过pytorch模型文件.pth、适合transformers库的模型文件.bin，而.gguf是适合llama.cpp的模型文件。将前两种模型转换为 GGML 模型，llama.cpp才能读取并使用此格式的模型进行推理。

```
python3 convert.py zh-models/7B/
```

我的执行：

```
!cd llama.cpp && python3 convert.py zh-models/7B/
---
Loading model file zh-models/7B/consolidated.00.pth
...
[291/291] Writing tensor output.weight | size 49954 x
4096 | type F16 | T+ 163
wrote zh-models/7B/ggml-model-f16.gguf
```

5、生成量化版本模型

进一步对FP16模型进行4-bit量化，生成量化模型文件路径为 zh-models/7B/ggml-model-q4_0.gguf，Q4_0代表量化方法

```
./quantize ./zh-models/7B/ggml-model-f16.gguf ./zh-models/7B/ggml-model-q4_0.gguf
```

我的执行：

```
!cd llama.cpp && ./quantize ./zh-models/7B/ggml-model-f16.gguf ./zh-models/7B/ggml-model-q4_0.gguf q4_0
```

6、加载并启动模型

运行`./main`二进制文件，`-m`命令指定GGML格式模型。以下是命令示例（并非最优参数）。

注意此命令来自Chinese-LLaMA-Alpaca仓库-wiki: <https://github.com/ymcui/Chinese-LLaMA-Alpaca/wiki/llama.cpp>, 相比llama.cpp官方仓库的readme: <https://github.com/ggerganov/llama.cpp?tab=readme-ov-file#run-the-quantized-model>

过时了。这里生成的不是.bin文件，而是.gguf文件

```
./main -m zh-models/7B/ggml-model-q4_0.bin --color -f prompts/alpaca.txt -ins -c 2048 --temp 0.2 -n 256 --repeat_penalty 1.1
```

在提示符`>`之后输入你的prompt，`cmd/ctrl+c`中断输出，多行信息以`\`作为行尾。如需查看帮助和参数说明，请执行`./main-h`命令。下面介绍一些常用的参数：

- c 控制上下文的长度，值越大越能参考更长的对话历史（默认：512）
- ins 启动类ChatGPT对话交流的instruction运行模式
- f 指定prompt模板，alpaca模型请加载prompts/alpaca.txt
- n 控制回复生成的最大长度（默认：128）
- b 控制batch size（默认：8），可适当增加
- t 控制线程数量（默认：4），可适当增加
- repeat_penalty 控制生成回复中对重复文本的惩罚力度
- temp 温度系数，值越低回复的随机性越小，反之越大
- top_p, top_k 控制解码采样的相关参数

```
./main -m ./models/mymodel/ggml-model-Q4_K_M.gguf -n 128
```

我的执行：

```
!cd llama.cpp && ./main -m ./zh-models/7B/ggml-model-q4_0.gguf --color -p "详细介绍一下北京的名胜古迹：" -n 256
---
Log start
main: build = 2515 (95562175)
main: built with cc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0 for x86_64-linux-gnu
main: seed = 1711231927
```

```
llama_model_loader: Loaded meta data with 19 key-value pairs and 291 tensors from
./zh-models/7B/ggml-model-q4_0.gguf (version GGUF V3 (latest))
llama_model_loader: Dumping metadata keys/values. Note: KV overrides do not apply
in this output.
llama_model_loader: - kv  0:                               general.architecture str
= llama
llama_model_loader: - kv  1:                               general.name str
= zh-models
llama_model_loader: - kv  2:                               llama.vocab_size u32
= 49954
llama_model_loader: - kv  3:                               llama.context_length u32
= 2048
llama_model_loader: - kv  4:                               llama.embedding_length u32
= 4096
llama_model_loader: - kv  5:                               llama.block_count u32
= 32
llama_model_loader: - kv  6:                               llama.feed_forward_length u32
= 11008
llama_model_loader: - kv  7:                               llama.rope.dimension_count u32
= 128
llama_model_loader: - kv  8:                               llama.attention.head_count u32
= 32
llama_model_loader: - kv  9:                               llama.attention.head_count_kv u32
= 32
llama_model_loader: - kv 10:                             llama.attention.layer_norm_rms_epsilon f32
= 0.000001
llama_model_loader: - kv 11:                               tokenizer.ggml.model str
= llama
llama_model_loader: - kv 13:                               tokenizer.ggml.tokens
arr[str,49954] = ["<unk>", "<s>", "</s>", "<0x00>", "<...>"]
llama_model_loader: - kv 14:                               tokenizer.ggml.scores
arr[f32,49954] = [0.000000, 0.000000, 0.000000, 0.0000...
llama_model_loader: - kv 15:                               tokenizer.ggml.token_type
arr[i32,49954] = [2, 3, 3, 6, 6, 6, 6, 6, 6, 6, 6, ...
llama_model_loader: - kv 16:                               tokenizer.ggml.add_bos_token bool
= true
llama_model_loader: - kv 17:                               tokenizer.ggml.add_eos_token bool
= false
llama_model_loader: - kv 18:                               general.quantization_version u32
= 2
llama_model_loader: - type  f32:   65 tensors
llama_model_loader: - type q4_0:  225 tensors
llama_model_loader: - type q6_K:   1 tensors
llm_load_vocab: mismatch in special tokens definition ( 581/49954 vs 259/49954 ).
llm_load_print_meta: format      = GGUF V3 (latest)
llm_load_print_meta: arch       = llama
llm_load_print_meta: vocab type = SPM
llm_load_print_meta: n_vocab    = 49954
llm_load_print_meta: n_merges   = 0
llm_load_print_meta: n_ctx_train = 2048
llm_load_print_meta: n_embd     = 4096
llm_load_print_meta: n_head     = 32
llm_load_print_meta: n_head_kv  = 32
llm_load_print_meta: n_layer    = 32
```

```
llm_load_print_meta: n_rot          = 128
llm_load_print_meta: n_embd_head_k  = 128
llm_load_print_meta: n_embd_head_v  = 128
llm_load_print_meta: n_gqa          = 1
llm_load_print_meta: n_embd_k_gqa   = 4096
llm_load_print_meta: n_embd_v_gqa   = 4096
llm_load_print_meta: f_norm_eps     = 0.0e+00
llm_load_print_meta: f_norm_rms_eps = 1.0e-06
llm_load_print_meta: f_clamp_kqv   = 0.0e+00
llm_load_print_meta: f_max_alibi_bias = 0.0e+00
llm_load_print_meta: f_logit_scale  = 0.0e+00
llm_load_print_meta: n_ff           = 11008
llm_load_print_meta: n_expert       = 0
llm_load_print_meta: n_expert_used  = 0
llm_load_print_meta: causal attn    = 1
llm_load_print_meta: pooling type   = 0
llm_load_print_meta: rope type      = 0
llm_load_print_meta: rope scaling    = linear
llm_load_print_meta: freq_base_train = 10000.0
llm_load_print_meta: freq_scale_train = 1
llm_load_print_meta: n_yarn_orig_ctx = 2048
llm_load_print_meta: rope_finetuned  = unknown
llm_load_print_meta: ssm_d_conv      = 0
llm_load_print_meta: ssm_d_inner     = 0
llm_load_print_meta: ssm_d_state     = 0
llm_load_print_meta: ssm_dt_rank     = 0
llm_load_print_meta: model type      = 7B
llm_load_print_meta: model ftype      = Q4_0
llm_load_print_meta: model params     = 6.89 B
llm_load_print_meta: model size       = 3.66 GiB (4.56 BPW)
llm_load_print_meta: general.name     = zh-models
llm_load_print_meta: BOS token        = 1 '<s>'
llm_load_print_meta: EOS token        = 2 '</s>'
llm_load_print_meta: UNK token        = 0 '<unk>'
llm_load_print_meta: LF token         = 13 '<0x0A>'

ggml_cuda_init: GGML_CUDA_FORCE_MMQ: no
ggml_cuda_init: CUDA_USE_TENSOR_CORES: yes
ggml_cuda_init: found 1 CUDA devices:
  Device 0: NVIDIA RTX A6000, compute capability 8.6, VMM: yes
llm_load_tensors: ggml ctx size = 0.11 MiB
llm_load_tensors: offloading 0 repeating layers to GPU
llm_load_tensors: offloaded 0/33 layers to GPU
llm_load_tensors: CPU buffer size = 3744.85 MiB
.....
.....
llama_new_context_with_model: n_ctx      = 512
llama_new_context_with_model: n_batch     = 512
llama_new_context_with_model: n_ubatch    = 512
llama_new_context_with_model: freq_base   = 10000.0
llama_new_context_with_model: freq_scale   = 1
llama_kv_cache_init: CUDA_Host KV buffer size = 256.00 MiB
llama_new_context_with_model: KV self size  = 256.00 MiB, K (f16): 128.00 MiB,
V (f16): 128.00 MiB
llama_new_context_with_model: CUDA_Host output buffer size = 97.57 MiB
llama_new_context_with_model: CUDA0 compute buffer size = 298.64 MiB
llama_new_context_with_model: CUDA_Host compute buffer size = 17.00 MiB
```

```

llama_new_context_with_model: graph nodes = 1060
llama_new_context_with_model: graph splits = 356

system_info: n_threads = 32 / 64 | AVX = 1 | AVX_VNNI = 0 | AVX2 = 1 | AVX512 = 1
| AVX512_VBMI = 0 | AVX512_VNNI = 1 | FMA = 1 | NEON = 0 | ARM_FMA = 0 | F16C = 1
| FP16_VA = 0 | WASM SIMD = 0 | BLAS = 1 | SSE3 = 1 | SSSE3 = 1 | VSX = 0 |
MATMUL_INT8 = 0 |

sampling:
    repeat_last_n = 64, repeat_penalty = 1.000, frequency_penalty = 0.000,
presence_penalty = 0.000
    top_k = 40, tfs_z = 1.000, top_p = 0.950, min_p = 0.050, typical_p = 1.000,
temp = 0.800
    mirostat = 0, mirostat_lr = 0.100, mirostat_ent = 5.000
sampling order:
CFG -> Penalties -> top_k -> tfs_z -> typical_p -> top_p -> min_p -> temperature
generate: n_ctx = 512, n_batch = 2048, n_predict = 256, n_keep = 1

```

详细介绍一下北京的名胜古迹：颐

```

llama_print_timings:      load time =   68652.39 ms
llama_print_timings:      sample time =       0.06 ms /     1 runs (  0.06 ms
per token, 18181.82 tokens per second)
llama_print_timings: prompt eval time =   70195.28 ms /    11 tokens ( 6381.39 ms
per token,  0.16 tokens per second)
llama_print_timings:      eval time =       0.00 ms /     1 runs (  0.00 ms
per token,      inf tokens per second)
llama_print_timings:      total time =  121061.36 ms /    12 tokens

```

发现非常之慢。一秒生成不出一个token

然后发现有人遇到了和我同样的问题：https://www.reddit.com/r/LocalLLaMA/comments/13xetjq/help_wanted_understanding_terrible_llamacpp_w/

解决方案：

该解决方案涉及传递特定的 `-t` (要使用的线程数量) 和 `-ng1` (要卸载的 GPU 层数量) 参数。

Here are the results for my machine:

以下是我的机器的结果：

command 命令	tokens/second 令牌/秒
<code>-ng1 2000000</code>	Less than 0.1 小于0.1
<code>-t 7</code>	1.6
<code>-t 7 -ng1 2000000</code>	8.5

Q：这两个参数是什么意思？

A：

<https://github.com/Eric-LLMs>

- `--threads N, -t N`: Set the number of threads to use during generation. Not used if model layers are offloaded to GPU. The server is using batching, this parameter is used only if one token is to be processed on CPU backend.
`--threads N, -t N`: 设置生成期间要使用的线程数。如果模型层卸载到 GPU，则不使用。服务器正在使用批处理，仅当要在 CPU 后端处理一个令牌时才使用此参数。
- `-ngl N, --n-gpu-layers N`: When compiled with appropriate support (currently CLBlast or cuBLAS), this option allows offloading some layers to the GPU for computation. Generally results in increased performance.
`-ngl N, --n-gpu-layers N`: 当使用适当的支持（当前是 CLBlast 或 cuBLAS）进行编译时，此选项允许将某些层卸载到 GPU 进行计算。通常会提高性能。

from <https://github.com/ggerganov/llama.cpp/tree/master/examples/server>

所以我只需要多加一个参数-t即可正常

```
!cd llama.cpp && ./main -m ./zh-models/7B/ggml-model-q4_0.gguf --color -p "详细介绍一下北京的名胜古迹: " -n 256 -t 7
---
详细介绍一下北京的名胜古迹: 故宫、天安门、颐和园、圆明园、长城。请注意用词准确, 不要含糊或使用夸张的语言。 [end of text]

llama_print_timings:      load time =    775.33 ms
llama_print_timings:      sample time =     2.12 ms /    33 runs ( 0.06 ms
per token, 15580.74 tokens per second)
llama_print_timings: prompt eval time =   983.58 ms /    11 tokens ( 89.42 ms
per token, 11.18 tokens per second)
llama_print_timings:      eval time =   3887.63 ms /    32 runs ( 121.49 ms
per token, 8.23 tokens per second)
llama_print_timings:      total time =  4889.66 ms /    43 tokens
Log end
```

再加一个参数-ngl 2000000, 就又成倍的快

```
!cd llama.cpp && ./main -m ./zh-models/7B/ggml-model-q4_0.gguf --color -p "详细介绍一下北京的名胜古迹: " -n 256 -t 7 -ngl 2000000
---
详细介绍一下北京的名胜古迹: 故宫、天安门、颐和园、圆明园、长城等。 [end of text]

llama_print_timings:      load time =    1417.98 ms
llama_print_timings:      sample time =     1.01 ms /    19 runs ( 0.05 ms
per token, 18905.47 tokens per second)
llama_print_timings: prompt eval time =   23.52 ms /    11 tokens ( 2.14 ms
per token, 467.69 tokens per second)
llama_print_timings:      eval time =   162.89 ms /    18 runs ( 9.05 ms
per token, 110.50 tokens per second)
llama_print_timings:      total time =  195.60 ms /    29 tokens
Log end
```

交互式执行

-c 控制上下文的长度，值越大越能参考更长的对话历史（默认：512）
-ins 启动类ChatGPT对话交流的instruction运行模式
-f 指定prompt模板，alpaca模型请加载prompts/alpaca.txt
-n 控制回复生成的最大长度（默认：128）
-b 控制batch size（默认：8），可适当增加
-t 控制线程数量（默认：4），可适当增加
--repeat_penalty 控制生成回复中对重复文本的惩罚力度
--temp 温度系数，值越低回复的随机性越小，反之越大
--top_p, top_k 控制解码采样的相关参数

```
# cd llama.cpp
# ./main -m ./zh-models/7B/ggml-model-q4_0.gguf --color -f prompts/alpaca.txt -ins -c 2048 --temp 0.2 -n 256 --repeat_penalty 1.1 -t 7 -ngl 2000000
```

> 介绍下北京

北京是中国的首都，也是一个历史悠久的城市。它位于中国北部，拥有丰富的文化遗产和现代化建筑。北京是一座具有悠久历史的城市，其历史可以追溯到公元前1045年。在明清时期（1368-1644），北京成为了中国的政治中心，并成为世界上最大的城市之一。

北京以其独特的文化和历史而闻名，其中包括紫禁城、天坛、长城等著名景点。此外，北京还是中国最大的商业和金融中心，拥有很多国际知名的公司和机构。

北京是一个现代化城市，其现代化建筑与传统文化相融合。它拥有世界一流的教育、医疗和科技设施，以及丰富的文化活动和艺术表演。

总的来说，北京是一个充满活力和历史的城市，以其独特的文化遗产、现代化建筑和现代化经济为特色而闻名于世。

> 北京有哪些著名建筑

北京有许多著名的建筑，其中包括：

1. 紫禁城：这是中国明清时期的皇宫，被誉为世界七大奇迹之一。
2. 天坛：这是中国古代的皇家祭天场所，也是一个重要的文化遗产。
3. 长城：这是世界上最长的城墙，建于明朝时期，现在是旅游胜地。
- 4.

> > 介绍下大模型

大模型是指一个大型的、高度复杂的人工智能系统。它通常由大量的数据和先进的算法构成，可以执行各种任务，如自然语言处理（NLP）、机器翻译（MT）、智能客服等。

大模型在许多领域都有广泛的应用，包括但不限于：

1. 自然语言处理：大模型可以用来理解和生成人类语言，例如回答问题、提供建议或生成文本。
2. 语音识别：大模型可以将语音转换为文本，并进行语音识别，以识别说话者的意图。
3. 智能客服：大模型可以与客户交互，通过自然语言处理技术回答用户的问题，并提供有用的信息和解决方案。
4. 机器翻译：大模型可以用来实现高质量的机器翻译，从一种语言自动生成另一种语言的文本。
5. 图像识别：大模型可以使用深度学习算法来理解和分析图像，以进行分类、识别物体或执行其他任务。

大模型具有高度的灵活性和适应性，能够根据不同的情境和需求进行自适应调整。它通过大量的数据训练，并利用强大的计算能力来处理复杂的问题。

总的来说，大模型是一种高度先进的人工智能

> 介绍下RAG

RAG（Reduce A Go）是一种游戏开发框架，用于构建和管理大型的多人在线游戏。

RAG提供了一个灵活、可扩展和易于使用的平台，可以帮助开发者创建复杂的游戏系统和功能。它支持多种编程语言和技术，包括C++、Python、JavaScript等，并提供丰富的API和工具，以简化开发过程。

RAG还具有高度的可定制性和自定义性，允许开发者根据自己的需求进行个性化设置。例如，开发者可以自定义游戏规则、图形引擎、物理引擎等，以满足特定的游戏风格或需求。

此外，RAG还提供了一个强大的社区支持系统，包括论坛、Issue Tracker和GitHub仓库等，以帮助开发人员解决遇到的问题和分享经验。

总的来说，RAG是一个功能强大且灵活的多人在线游戏开发框架，可以为开发者提供一个高效、可扩展和易于使用的平台。

```
llama_print_timings:      load time =    1566.37 ms
llama_print_timings:      sample time =   1628.49 ms /  2022 runs ( 0.81 ms
per token, 1241.64 tokens per second)
llama_print_timings: prompt eval time = 201680.04 ms /  357 tokens ( 564.93 ms
per token, 1.77 tokens per second)
llama_print_timings:      eval time =   33499.21 ms /  2008 runs ( 16.68 ms
per token, 59.94 tokens per second)
llama_print_timings:      total time = 350632.16 ms /  2365 tokens
```

7、评估模型

参考：<https://github.com/ggerganov/llama.cpp?tab=readme-ov-file#perplexity-measuring-model-quality>

perplexity困惑度，可以衡量模型质量，困惑度越低越好

用量化后的模型跑一遍这个测试集：[wikitext](#)

8、部署模型

HTTP server HTTP服务器

[llama.cpp web server](#) is a lightweight [OpenAI API](#) compatible HTTP server that can be used to serve local models and easily connect them to existing clients.

llama.cpp Web 服务器是一个轻量级 OpenAI API 兼容的 HTTP 服务器，可用于服务本地模型并轻松将它们连接到现有客户端。

linux下

```
./server -m models/7B/ggml-model.gguf -c 2048
```

The above command will start a server that by default listens on `127.0.0.1:8080`. You can consume the endpoints with Postman or NodeJS with axios library. You can visit the web front end at the same url.

上面的命令将启动一个默认侦听 `127.0.0.1:8080` 的服务器。您可以使用带有 axios 库的 Postman 或 NodeJS 来使用端点。您可以通过相同的 URL 访问 Web 前端。

我的执行：

```
./server -m ./zh-models/7B/ggml-model-q4_0.gguf -c 2048 -t 7 -ngl 2000000
```

显示：

```
root@711bb883812f:~/llama.cpp# ./server -m ./zh-models/7B/ggml-model-q4_0.gguf -c 2048 -t 7 -ngl 2000000
{"tid":"139905957322752","timestamp":1711237655,"level":"INFO","function":"main",
"line":2781,"msg":"build info","build":2515,"commit":"95562175"}
 {"tid":"139905957322752","timestamp":1711237655,"level":"INFO","function":"main",
"line":2786,"msg":"system
info","n_threads":7,"n_threads_batch":-1,"total_threads":64,"system_info":"AVX =
1 | AVX_VNNI = 0 | AVX2 = 1 | AVX512 = 1 | AVX512_VBMI = 0 | AVX512_VNNI = 1 |
FMA = 1 | NEON = 0 | ARM_FMA = 0 | F16C = 1 | FP16_VA = 0 | WASM SIMD = 0 | BLAS
= 1 | SSE3 = 1 | SSSE3 = 1 | VSX = 0 | MATMUL_INT8 = 0 | "}
llama_model_loader: loaded meta data with 19 key-value pairs and 291 tensors from
./zh-models/7B/ggml-model-q4_0.gguf (version GGUF V3 (latest))
llama_model_loader: Dumping metadata keys/values. Note: KV overrides do not apply
in this output.
llama_model_loader: - kv  0:                               general.architecture str
= llama
llama_model_loader: - kv  1:                               general.name str
= zh-models
llama_model_loader: - kv  2:                               llama.vocab_size u32
= 49954
llama_model_loader: - kv  3:                               llama.context_length u32
= 2048
llama_model_loader: - kv  4:                               llama.embedding_length u32
= 4096
llama_model_loader: - kv  5:                               llama.block_count u32
= 32
llama_model_loader: - kv  6:                               llama.feed_forward_length u32
= 11008
llama_model_loader: - kv  7:                               llama.rope.dimension_count u32
= 128
llama_model_loader: - kv  8:                               llama.attention.head_count u32
= 32
llama_model_loader: - kv  9:                               llama.attention.head_count_kv u32
= 32
llama_model_loader: - kv 10:                               llama.attention.layer_norm_rms_epsilon f32
= 0.000001
llama_model_loader: - kv 11:                               general.file_type u32
= 2
llama_model_loader: - kv 12:                               tokenizer.ggml.model str
= llama
llama_model_loader: - kv 13:                               tokenizer.ggml.tokens
arr[str,49954] = [<unk>, <s>, </s>, <0x00>, <...
llama_model_loader: - kv 14:                               tokenizer.ggml.scores
arr[f32,49954] = [0.000000, 0.000000, 0.000000, 0.0000...
llama_model_loader: - kv 15:                               tokenizer.ggml.token_type
arr[i32,49954] = [2, 3, 3, 6, 6, 6, 6, 6, 6, 6, ...
llama_model_loader: - kv 16:                               tokenizer.ggml.add_bos_token bool
= true
```

```
llama_model_loader: - kv 17:           tokenizer.ggml.add_eos_token bool  
= false  
llama_model_loader: - kv 18:           general.quantization_version u32  
= 2  
llama_model_loader: - type f32:   65 tensors  
llama_model_loader: - type q4_0:  225 tensors  
llama_model_loader: - type q6_K:   1 tensors  
llm_load_vocab: mismatch in special tokens definition ( 581/49954 vs 259/49954 ).  
llm_load_print_meta: format      = GGUF V3 (latest)  
llm_load_print_meta: arch       = llama  
llm_load_print_meta: vocab type = SPM  
llm_load_print_meta: n_vocab    = 49954  
llm_load_print_meta: n_merges   = 0  
llm_load_print_meta: n_ctx_train= 2048  
llm_load_print_meta: n_embd     = 4096  
llm_load_print_meta: n_head     = 32  
llm_load_print_meta: n_head_kv  = 32  
llm_load_print_meta: n_layer    = 32  
llm_load_print_meta: n_rot      = 128  
llm_load_print_meta: n_embd_head_k= 128  
llm_load_print_meta: n_embd_head_v= 128  
llm_load_print_meta: n_gqa      = 1  
llm_load_print_meta: n_embd_k_gqa= 4096  
llm_load_print_meta: n_embd_v_gqa= 4096  
llm_load_print_meta: f_norm_eps = 0.0e+00  
llm_load_print_meta: f_norm_rms_eps= 1.0e-06  
llm_load_print_meta: f_clamp_kqv = 0.0e+00  
llm_load_print_meta: f_max_alibi_bias= 0.0e+00  
llm_load_print_meta: f_logit_scale= 0.0e+00  
llm_load_print_meta: n_ff       = 11008  
llm_load_print_meta: n_expert   = 0  
llm_load_print_meta: n_expert_used= 0  
llm_load_print_meta: causal attn = 1  
llm_load_print_meta: pooling type= 0  
llm_load_print_meta: rope type  = 0  
llm_load_print_meta: rope scaling= linear  
llm_load_print_meta: freq_base_train= 10000.0  
llm_load_print_meta: freq_scale_train= 1  
llm_load_print_meta: n_yarn_orig_ctx= 2048  
llm_load_print_meta: rope_finetuned= unknown  
llm_load_print_meta: ssm_d_conv  = 0  
llm_load_print_meta: ssm_d_inner = 0  
llm_load_print_meta: ssm_d_state = 0  
llm_load_print_meta: ssm_dt_rank= 0  
llm_load_print_meta: model type = 7B  
llm_load_print_meta: model ftype = Q4_0  
llm_load_print_meta: model params= 6.89 B  
llm_load_print_meta: model size  = 3.66 GiB (4.56 BPW)  
llm_load_print_meta: general.name= zh-models  
llm_load_print_meta: BOS token  = 1 '<s>'  
llm_load_print_meta: EOS token  = 2 '</s>'  
llm_load_print_meta: UNK token  = 0 '<unk>'  
llm_load_print_meta: LF token   = 13 '<0x0A>'  
ggml_cuda_init: GGML_CUDA_FORCE_MMQ: no  
ggml_cuda_init: CUDA_USE_TENSOR_CORES: yes  
ggml_cuda_init: found 1 CUDA devices:
```

```

Device 0: NVIDIA RTX A6000, compute capability 8.6, VMM: yes
llm_load_tensors: ggml ctx size = 0.22 MiB
llm_load_tensors: offloading 32 repeating layers to GPU
llm_load_tensors: offloading non-repeating layers to GPU
llm_load_tensors: offloaded 33/33 layers to GPU
llm_load_tensors: CPU buffer size = 109.76 MiB
llm_load_tensors: CUDA0 buffer size = 3635.09 MiB

.....
.....
llama_new_context_with_model: n_ctx      = 2048
llama_new_context_with_model: n_batch     = 2048
llama_new_context_with_model: n_ubatch   = 512
llama_new_context_with_model: freq_base  = 10000.0
llama_new_context_with_model: freq_scale = 1
llama_kv_cache_init:      CUDA0 KV buffer size = 1024.00 MiB
llama_new_context_with_model: KV self size = 1024.00 MiB, K (f16): 512.00 MiB,
v (f16): 512.00 MiB
llama_new_context_with_model: CUDA_Host output buffer size = 390.27 MiB
llama_new_context_with_model:      CUDA0 compute buffer size = 164.00 MiB
llama_new_context_with_model: CUDA_Host compute buffer size = 12.00 MiB
llama_new_context_with_model: graph nodes = 1060
llama_new_context_with_model: graph splits = 2
{"tid":"139905957322752","timestamp":1711237657,"level":"INFO","function":"init",
"line":702,"msg":"initializing slots","n_slots":1}
 {"tid":"139905957322752","timestamp":1711237657,"level":"INFO","function":"init",
"line":711,"msg":"new slot","id_slot":0,"n_ctx_slot":2048}
 {"tid":"139905957322752","timestamp":1711237657,"level":"INFO","function":"main",
"line":2881,"msg":"model loaded"}
 {"tid":"139905957322752","timestamp":1711237657,"level":"INFO","function":"main",
"line":2903,"msg":"chat template","chat_example":"<|im_start|>system\nYou are a
helpful
assistant<|im_end|>\n<|im_start|>user\nHello<|im_end|>\n<|im_start|>assistant\nHi
there<|im_end|>\n<|im_start|>user\nHow are you?
<|im_end|>\n<|im_start|>assistant\n","built_in":true}
 {"tid":"139905957322752","timestamp":1711237657,"level":"INFO","function":"main",
"line":3524,"msg":"HTTP server
listening","n_threads_http":"63","port":"8080","hostname":"127.0.0.1"}
 {"tid":"139905957322752","timestamp":1711237657,"level":"INFO","function":"update
_slots","line":1667,"msg":"all slots are idle"}

```

CURL测试：

```

curl --request POST \
--url http://localhost:8080/completion \
--header "Content-Type: application/json" \
--data '{"prompt": "Building a website can be done in 10 simple
steps:","n_predict": 128}'

```

成功返回：

```
root@71bb883812f:~/llama.cpp# curl --request POST \
--url http://localhost:8080/completion \
--header "Content-Type: application/json" \
--data '{"prompt": "Building a website can be done in 10 simple
steps:","n_predict": 128}'
{"content":" Choose a domain name, select a web hosting service, design a
website, add content to your website, create a Contact Us page, design a shopping
cart, add a search engine optimization (SEO) strategy, create a website analytics
system, and create a website maintenance schedule. 步骤 7： 创建一个 Contact Us 页面
： 创建一个 Contact Us 页面，用于接收客户的联系信息，包括姓名、电子邮件、电话号码和其他必要
信息。 创建一个 Contact Us 页面，可以帮助你的客户快速找到想要的联系信息。 创建",
"id_slot":0,"stop":true,"model": "./zh-models/7B/ggml-model-
q4_0.gguf","tokens_predicted":128,"tokens_evaluated":12,"generation_settings":
{"n_ctx":2048,"n_predict": -1,"model": "./zh-models/7B/ggml-model-
q4_0.gguf","seed":4294967295,"temperature":0.800000011920929,"dynatemp_range":0.0
,"dynatemp_exponent":1.0,"top_k":40,"top_p":0.949999988079071,"min_p":0.050000000
74505806,"tfs_z":1.0,"typical_p":1.0,"repeat_last_n":64,"repeat_penalty":1.0,"pre
sence_penalty":0.0,"frequency_penalty":0.0,"penalty_prompt_tokens":
[],"use_penalty_prompt_tokens":false,"mirostat":0,"mirostat_tau":5.0,"mirostat_eta":0.10000000149011612,"penalize_n1":false,"stop":
[],"n_keep":0,"ignore_eos":false,"stream":false,"logit_bias":
[],"n_probs":0,"min_keep":0,"grammar":"","samplers":
["top_k","tfs_z","typical_p","top_p","min_p","temperature"]},"prompt":"Building a
website can be done in 10 simple
steps:","truncated":false,"stopped_eos":false,"stopped_word":false,"stopped_limit
":true,"stopping_word": "", "tokens_cached":139,"timings":
{"prompt_n":12,"prompt_ms":50.152,"prompt_per_token_ms":4.179333333333333,"prompt
_per_second":239.27261126176424,"predicted_n":128,"predicted_ms":1195.021,"predic
ted_per_token_ms":9.3361015625,"predicted_per_second":107.11108842438753}}root@71
1bb883812f:~/llama.cpp#
```

```
curl --request POST \
--url http://localhost:8080/completion \
--header "Content-Type: application/json" \
--data '{"prompt": "建立一个网站可以通过10个简单的步骤来完成: ","n_predict": 128}'
```

微调LLM模型之Huatuo

官方仓库: [Huatuo-Llama-Med-Chinese](#)

官方文章:

[2023/09/24]发布《面向智慧医疗的大语言模型微调技术》

[2023/09/12]在arxiv发布《探索大模型从医学文献中交互式知识的获取》

[2023/09/08]在arxiv发布《基于知识微调的大语言模型可靠中文医学回复生成方法》

项目简介

本草（原名：华驼）模型仓库，基于中文医学知识的大语言模型指令微调

本项目开源了经过中文医学指令精调/指令微调(Instruction-tuning) 的大语言模型集，包括 LLaMA、Alpaca-Chinese、Bloom、活字模型等。

我们基于医学知识图谱以及医学文献，结合ChatGPT API构建了中文医学指令微调数据集，并以此对各种基模型进行了指令微调，提高了基模型在医疗领域的问答效果。

我们想使用的：

[2023/03/31] 增加了基于LLaMA和医学知识库进行指令微调的模型发布。

1、克隆仓库

```
git clone https://github.com/SCIR-HI/Huatuo-Llama-Med-Chinese.git
```

2、安装依赖

查看所需依赖，这里面的不用全部安装

```
root@608dbef3510b:~/Huatuo-Llama-Med-Chinese# cat requirements.txt
accelerate==0.20.1
appdirs==1.4.4
bitsandbytes==0.37.2
black==23.3.0
black[jupyter]==23.3.0
datasets==2.11.0
fire==0.5.0
peft==0.3.0
transformers==4.30.1
gradio==3.33.1
sentencepiece==0.1.97
scipy==1.10.1
wandb
```

基于前面【运行LLaMA模型/4安装依赖】的安装，只安装这些就行：

```
pip install fire==0.6.0
# Successfully installed fire-0.6.0 termcolor-2.4.0

pip install gradio==3.33.1
# Successfully installed aiofiles-23.2.1 aiohttp-3.9.3 aiosignal-1.3.1 altair
5.2.0 asyncio-timeout-4.0.3 contourpy-1.2.0 cycler-0.12.1 fastapi-0.110.0 ffmpy
0.3.2 fonttools-4.50.0 frozenlist-1.4.1 gradio-3.33.1 gradio-client-0.13.0 h11
0.14.0 httpcore-1.0.4 httpx-0.27.0 kiwisolver-1.4.5 linkify-it-py-2.0.3 markdown
it-py-2.2.0 matplotlib-3.8.3 mdit-py-plugins-0.3.3 mdurl-0.1.2 multidict-6.0.5
orjson-3.9.15 pydub-0.25.1 pyparsing-3.1.2 python-multipart-0.0.9 semantic-
version-2.10.0 starlette-0.36.3 toolz-0.12.1 uc-micro-py-1.0.3 uvicorn-0.29.0
websockets-11.0.3 yarl-1.9.4
```

3、LoRA模型权重下载

LoRA权重可以通过百度网盘或Hugging Face下载：

4. 对LLaMA进行指令微调的LoRA权重文件
 - 基于医学知识库 [百度网盘](#)和[Hugging Face](#)

git下载：

```
git clone https://huggingface.co/thinksoso/lora-llama-med
```

我的执行：

```
root@608dbef3510b:~# cd models

root@608dbef3510b:~/models# ls
chinese_llama_plus_lora_7b      chinese_llama_plus_lora_7b_merge_model  llama
                               llama-7b-hf-1
chinese_llama_plus_lora_7b.zip  download_llama_only_7b.sh          llama-7b-
hf

root@608dbef3510b:~/models# mkdir huatuo

root@608dbef3510b:~/models# cd huatuo

root@608dbef3510b:~/models/huatuo# git clone
https://huggingface.co/thinksoso/lora-llama-med
Cloning into 'lora-llama-med'...
remote: Enumerating objects: 17, done.
remote: Total 17 (delta 0), reused 0 (delta 0), pack-reused 17

Unpacking objects: 100% (17/17), 1.95 KiB | 665.00 KiB/s, done.

root@608dbef3510b:~/models/huatuo# ls lora-llama-med/
README.md  adapter_config.json  adapter_model.bin
```

█ / ... / huatuo / lora-llama-med /

Name	Last Modified
adapter_config.json	2 days ago
adapter_model.bin	2 days ago
README.md	2 days ago

4、推理微调好的模型

infer.sh脚本代码如下，请将下列代码中基模型base_model、lora权重lora_weights以及测试数据集路径instruct_dir进行替换后运行

```
python infer.py \
    --base_model 'BASE_MODEL_PATH' \
    --lora_weights 'LORA_WEIGHTS_PATH' \
    --use_lora True \
    --instruct_dir 'INFER_DATA_PATH' \
    --prompt_template 'TEMPLATE_PATH'
```

官方示例

```
python infer.py \
    --base_model 'decapoda-research/llama-7b-hf' \
    --lora_weights './lora-llama-med' \
    --use_lora True \
    --instruct_dir './data/infer.json' \
    --prompt_template 'med_template'
```

data/infer.json

```
{"instruction": "小张最近感觉身体不适，出现心悸、气促等症状。体检发现心脏扩大、搏动减弱。", "input": "", "output": "小张可能患有心肌炎，建议进行心电图和心脏超声等检查来确定诊断。治疗方案包括使用泼尼松、生脉饮和三磷酸腺苷等药物，同时建议适当控制体温，保持良好的营养状况。"}, {"instruction": "小王被确诊为肝炎双重感染，最可能的并发症是什么？", "input": "", "output": "最可能的并发症是肝癌和肝硬化。"}, {"instruction": "麻风病和儿童哮喘的病因是否一致？", "input": "", "output": "不一致，麻风病的病因是麻风分枝杆菌，而儿童哮喘的病因是气候、药物、吸入过敏原等。"}, {"instruction": "一位患者出现了持续性的胀痛和黄疸，经影像学检查后确诊为肝胆管结石病，应该采取哪些治疗方案？", "input": "", "output": "可以考虑采用残石的处理、微创保肝取石术或手术治疗等治疗方案。"}, {"instruction": "一位35岁女性患者出现了关节疼痛和糖耐量异常，请问她可能患了什么疾病？", "input": "", "output": "该患者可能患上了慢性自身免疫性胰腺炎，伴有慢性风湿性关节炎和糖耐量异常的症状。建议到消化内科进行检查和治疗。"}
```

我的执行：

```
cd Huatuo-Llama-Med-Chinese

python infer.py \
    --base_model '/home/models/llama-7b-hf' \
    --lora_weights '/home/models/huatuo/lora-llama-med' \
    --use_lora True \
    --instruct_dir './data/infer.json' \
    --prompt_template 'med_template'
```

查看结果

```
root@608dbef3510b:/Huatuo-Llama-Med-Chinese# python infer.py      --base_model '/home/models/llama-7b-hf'      --lora_weights '/home/m
odels/huatuo/lora-llama-med'      --use_lora True      --instruct_dir './data/infer.json'      --prompt_template 'med_template'
/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/gradio_client/documentation.py:103: UserWarning: Could not get documentation group for <class 'gradio.mix.Parallel'>: No known documentation group for module 'gradio.mix'
warnings.warn(f"Could not get documentation group for {cls}: {exc}")
/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/gradio_client/documentation.py:103: UserWarning: Could not get documentation group for <class 'gradio.mix.Series'>: No known documentation group for module 'gradio.mix'
warnings.warn(f"Could not get documentation group for {cls}: {exc}")
Loading checkpoint shards: 100% |██████████| 2/2 [00:47<00:00, 23.68s/it]
using lora /home/models/huatuo/lora-llama-med
###infering###
###instruction###
小张最近感觉身体不适，出现心悸、气促等症状。体检发现心脏扩大、搏动减弱。
###golden output###
小张可能患有心肌炎，建议进行心电图和心脏超声等检查来确定诊断。治疗方案包括使用泼尼松、生脉饮和三磷酸腺苷等药物，同时建议适当控制体温，保持良好的营养状况。
###model output###
小张可能患有心肌炎，建议尽快到医院进行检查和治疗。治疗方案包括药物治疗和手术治疗。</s>
###infering###
###instruction###
小王被确诊为肝炎双重感染，最可能的并发症是什么？
###golden output###
最可能的并发症是肝癌和肝硬化。
###model output###
肝炎双重感染的最可能的并发症包括肝硬化、肝炎、感染等。</s>
###infering###
###instruction###
麻风病和儿童哮喘的病因是否一致？
###golden output###
不一致。麻风病的病因是麻风分枝杆菌，而儿童哮喘的病因是气候、药物、吸入过敏原等。
###model output###
不一致。麻风病的病因是麻风感染，而儿童哮喘的病因是喂养不良、肺炎、呼吸道感染等多种因素。</s>
###infering###
###instruction###
一位患者出现了持续性的胀痛和黄疸，经影像学检查后确诊为胆管结石病，应该采取哪些治疗方案？
###golden output###
可以考虑采用结石的处理、微创保肝取石术或手术治疗等治疗方案。
###model output###
针对胆管结石病，常用的治疗方案包括药物治疗、手术治疗和辅助治疗等。具体治疗方案需要根据患者的具体情况而定。</s>
###infering###
###instruction###
一位35岁女性患者出现了关节疼痛和糖耐量异常，请问她可能患了什么疾病？
###golden output###
该患者可能患上了慢性自身免疫性胰腺炎，伴有慢性风湿性关节炎和糖耐量异常的症状。建议到消化内科进行检查和治疗。
###model output###
根据症状描述，该患者可能患有胰腺炎。建议尽快到内科进行检查和治疗。</s>
root@608dbef3510b:/Huatuo-Llama-Med-Chinese#
```

推理infer.py里写好的几个测试用例

```
python infer.py \
    --base_model '/home/models/llama-7b-hf' \
    --lora_weights '/home/models/huatuo/lora-llama-med' \
    --use_lora True \
    --prompt_template 'med_template'
```

5、准备微调

准备数据

https://raw.githubusercontent.com/SCIR-HI/Huatuo-Llama-Med-Chinese/main/data/llama_data.json

8000余条

{"instruction": "一名年龄在70岁的女性，出现了晕厥、不自主颤抖、情绪不稳等症状，请详细说明其手术治疗和术前准备。", "input": "", "output": "该病需要进行电极导线、脉冲发生器和永久心脏起搏器置入术，并需要使用镇静药物和局麻对病人进行手术治疗。术前准备包括1-3天的时间进行术前检查和生活方式的调整。"}

{"instruction": "一个患者的卵巢小细胞癌转移至其它部位，是否有必要进行手术治疗？", "input": "", "output": "当卵巢小细胞癌转移至其它部位时，手术治疗的效果可能不理想，因此一般不推荐进行手术治疗。针对转移病灶，可以采用化疗、放疗等治疗手段进行综合治疗。"}

{"instruction": "一位60岁男性患者由于肝动脉瘤出现肝功能异常，具体表现为黄疸，该如何诊治？", "input": "", "output": "该患者可能出现了肝动脉瘤并发症之一的梗阻性黄疸，需要进行腹部平片、彩色多普勒超声、CTA、动脉造影等检查后确定诊断，治疗方面应按照病因治疗，如若为肝动脉炎、动脉粥样硬化等病因，应该采取手术治疗，而若是单纯的肝动脉瘤，可以进行介入治疗或手术治疗。"}

{"instruction": "患者男，28岁，关节部位红肿疼痛，排尿困难，近期有过微生物感染史。请问可能患的是什么疾病？如何治疗？", "input": "", "output": "可能患有反应性关节炎。治疗方案可以采用沙利度胺、泼尼松等药物。同时还要注意休息和饮食。"}

改写执行命令

查看 ./scripts/finetune.sh 提供的写法：

```
exp_tag="e1"
python finetune.py \
    --base_model 'decapoda-research/llama-7b-hf' \
    --data_path './data/llama_data.json' \
    --output_dir './lora-llama-med-'$exp_tag' \
    --prompt_template_name 'med_template' \
    --micro_batch_size 128 \
    --batch_size 128 \
    --wandb_run_name $exp_tag
```

改写一下：

创建lora-llama-med-e1文件夹

```
mkdir lora-llama-med-e1
```

我们真正运行的命令

```
python finetune.py \
    --base_model '/home/models/llama-7b-hf' \
    --data_path './data/llama_data.json' \
    --output_dir './lora-llama-med-e1' \
    --prompt_template_name 'med_template' \
    --micro_batch_size 128 \
    --batch_size 128 \
    --wandb_run_name e1
```

安装依赖

想执行微调命令还要继续安装几个包

```
pip install wandb
# Successfully installed GitPython-3.1.42 appdirs-1.4.4 docker-pycreds-0.4.0
gitdb-4.0.11 sentry-sdk-1.42.0 setproctitle-1.3.3 smmap-5.0.1 wandb-0.16.4

pip install datasets
# Successfully installed datasets-2.18.0 dill-0.3.8 multiprocess-0.70.16 pyarrow-
15.0.2 pyarrow-hotfix-0.6 xxhash-3.4.1

# ImportError: Using `load_in_8bit=True` requires Accelerate: `pip install
accelerate` and the latest version of bitsandbytes `pip install -i
https://test.pypi.org/simple/ bitsandbytes` or pip install bitsandbytes` 

# pip install bitsandbytes==0.37.2
# requirements.txt里提到的这个包版本有bug
# 需要用0.43.0的版本
pip install bitsandbytes==0.43.0
# Successfully installed bitsandbytes-0.43.0
```

wandb教程

<https://wandb.ai/site>

wandb login

<https://docs.wandb.ai/quickstart>

学习教程：使用wandb可视化训练过程运行该教程，得到wandb的输出

ember404 > Projects > my-awesome-project

Invite Teammates

Ember Personal

Autosaved 2 minutes ago

No metrics logged yet.

Charts will appear here as data is logged.

Create report

ember404 > Projects > my-awesome-project > Runs > curious-gorge-2 > Workspace

Invite Teammates

Ember Personal

Autosaved just now

accuracy

loss

System 18

GPU Power Usage (W)

GPU Power Usage (%)

GPU Memory Allocated (%)

GPU Time Spent Accessing Memory (%)

GPU Temp (°C)

Add panel

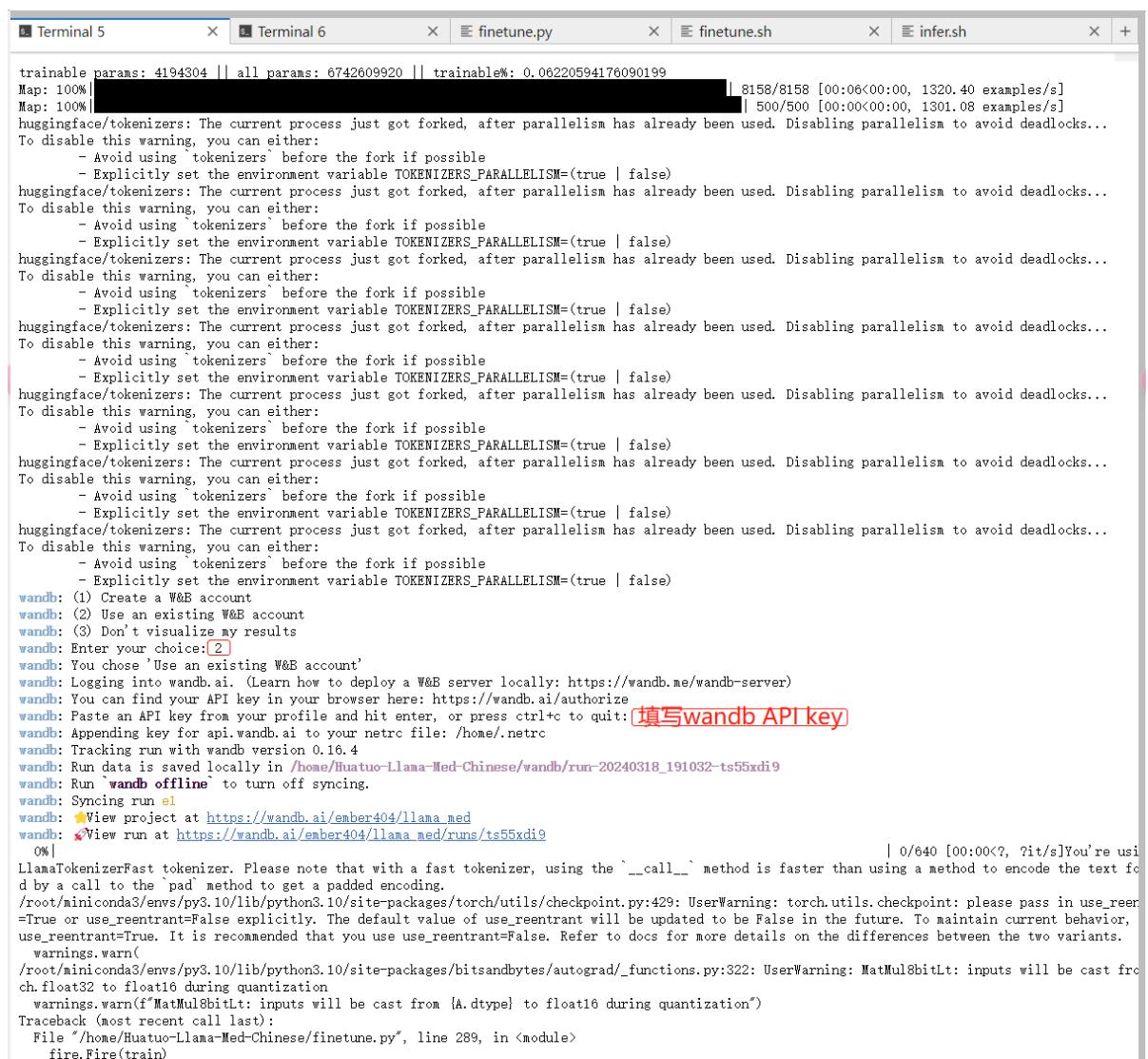
6、开始微调

第一次运行，失败

失败了，batch_size设置的过大，显存不够

```
root@608dbef3510b:~/Huatuo-Llama-Med-Chinese#
root@608dbef3510b:~/Huatuo-Llama-Med-Chinese# python finetune.py --base_model '/home/models/llama-7b-hf' --data_path './data/llama_data.json' --output_dir './lora-llama-med-el' --prompt_template_name 'med_template' --micro_batch_size 128 --batch_size 128 --wandb_run_name el
Training Alpaca-LoRA model with params:
base_model: /home/models/llama-7b-hf
data_path: ./data/llama_data.json
output_dir: ./lora-llama-med-el
batch_size: 128
micro_batch_size: 128
num_epochs: 10
learning_rate: 0.0003
cutoff_len: 256
val_set_size: 500
lora_r: 8
lora_alpha: 16
lora_dropout: 0.05
lora_target_modules: ['q_proj', 'v_proj']
train_on_inputs: False
group_by_length: False
wandb_project: llama_med
wandb_run_name: el
wandb_watch:
wandb_log_model:
resume_from_checkpoint: False
prompt template: med_template

Loading checkpoint shards: 100%|██████████| 2/2 [00:14<00:00, 7.03s/it]
Generating train split: 8658 examples [00:00, 344009.06 examples/s]
trainable params: 4194304 || all params: 6742609920 || trainable%: 0.06220594176090199
Map: 100%|██████████| 8158/8158 [00:06<00:00, 1320.40 examples/s]
Map: 100%|██████████| 500/500 [00:00<00:00, 1301.08 examples/s]
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
```



```
trainable params: 4194304 || all params: 6742609920 || trainable%: 0.06220594176090199
Map: 100%|██████████| 8158/8158 [00:06<00:00, 1320.40 examples/s]
Map: 100%|██████████| 500/500 [00:00<00:00, 1301.08 examples/s]
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
wandb: (1) Create a W&B account
wandb: (2) Use an existing W&B account
wandb: (3) Don't visualize my results
wandb: Enter your choice:[2]
wandb: You chose 'Use an existing W&B account'
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can view your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit: (填写wandb API key)
wandb: Appending key for api.wandb.ai to your netrc file: /home/.netrc
wandb: Tracking run with wandb version 0.16.4
wandb: Run data is saved locally in /home/Huatuo-Llama-Med-Chinese/wandb/run-20240318_191032-ts55xd19
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run el
wandb: ★View project at https://wandb.ai/ember404/llama\_med
wandb: ★View run at https://wandb.ai/ember404/llama\_med/runs/ts55xd19
  0%|          | 0/640 [00:00<?, ?it/s]You're using LLaMA TokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using a method to encode the text fed by a call to the `pad` method to get a padded encoding.
/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/utils/checkpoint.py:429: UserWarning: torch.utils.checkpoint: please pass in use_reentrant=True or use_reentrant=False explicitly. The default value of use_reentrant will be updated to be False in the future. To maintain current behavior, use_reentrant=True. It is recommended that you use use_reentrant=False. Refer to docs for more details on the differences between the two variants.
  warnings.warn(
/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/bitsandbytes/autograd/_functions.py:322: UserWarning: MatMul8bitLt: inputs will be cast from float32 to float16 during quantization
  warnings.warn("MatMul8bitLt: inputs will be cast from {A.dtype} to float16 during quantization")
Traceback (most recent call last):
  File "/home/Huatuo-Llama-Med-Chinese/fine_tune.py", line 289, in <module>
    fire.Fire(train)
```

```

File "/home/Huatuo-Llama-Med-Chinese/finetune.py", line 289, in <module>
    fire.Fire(train)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py", line 143, in Fire
    component_trace = _Fire(component, args, parsed_flag_args, context, name)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py", line 477, in _Fire
    component, remaining_args = _CallAndUpdateTrace(
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py", line 693, in _CallAndUpdateTrace
    component = fn(**varargs, **kwargs)
File "/home/Huatuo-Llama-Med-Chinese/finetune.py", line 279, in train
    trainer.train(resume_from_checkpoint=resume_from_checkpoint)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/training/trainer.py", line 1662, in train
    return inner_training_loop()
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/training/trainer.py", line 1929, in _inner_training_loop
    tr_loss_step = self.training_step(model, inputs)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/training/trainer.py", line 2709, in training_step
    self.scaler.scale(loss).backward()
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/_tensor.py", line 492, in backward
    torch.autograd.backward()
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/autograd/__init__.py", line 251, in backward
    Variable._execution_engine.run_backward( # Calls into the C++ engine to run the backward pass
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/autograd/function.py", line 288, in apply
    return user_fn(self, *args)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/utils/checkpoint.py", line 288, in backward
    torch.autograd.backward(outputs_with_grad, args_with_grad)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/autograd/__init__.py", line 251, in backward
    Variable._execution_engine.run_backward( # Calls into the C++ engine to run the backward pass
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 1.34 GiB. GPU 0 has a total capacity of 47.54 GiB of which 857.00 MiB is free. Process 3124114 has 46.68 GiB memory in use. Of the allocated memory 39.27 GiB is allocated by PyTorch, and 7.08 GiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting max_split_size_mb to avoid fragmentation. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
Traceback (most recent call last):
  File "/home/Huatuo-Llama-Med-Chinese/finetune.py", line 289, in <module>
    fire.Fire(train)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py", line 143, in Fire
    component_trace = _Fire(component, args, parsed_flag_args, context, name)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py", line 477, in _Fire
    component, remaining_args = _CallAndUpdateTrace(
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py", line 693, in _CallAndUpdateTrace
    component = fn(**varargs, **kwargs)
File "/home/Huatuo-Llama-Med-Chinese/finetune.py", line 279, in train
    trainer.train(resume_from_checkpoint=resume_from_checkpoint)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/training/trainer.py", line 1662, in train
    return inner_training_loop()
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/training/trainer.py", line 1929, in _inner_training_loop
    tr_loss_step = self.training_step(model, inputs)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/training/trainer.py", line 2709, in training_step
    self.scaler.scale(loss).backward()
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/_tensor.py", line 492, in backward
    torch.autograd.backward()
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/autograd/__init__.py", line 251, in backward
    Variable._execution_engine.run_backward( # Calls into the C++ engine to run the backward pass
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/autograd/function.py", line 288, in apply
    return user_fn(self, *args)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/utils/checkpoint.py", line 288, in backward
    torch.autograd.backward(outputs_with_grad, args_with_grad)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/autograd/__init__.py", line 251, in backward
    Variable._execution_engine.run_backward( # Calls into the C++ engine to run the backward pass
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 47.54 GiB. GPU 0 has a total capacity of 47.54 GiB of which 857.00 MiB is free. Process 3124114 has 46.68 GiB memory in use. Of the allocated memory 39.27 GiB is allocated by PyTorch, and 7.08 GiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting max_split_size_mb to avoid fragmentation. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
wandb: View run e1 at: https://wandb.ai/ember404/llama_med/runs/ts55xd19
wandb: View job at https://wandb.ai/ember404/llama_med/jobs/QXJ0aWZhY3RDb2xsZWN0aW9uOE1MDA5NDQ0NA==/version_details/v0
wandb: Synced 6 W&B file(s), 0 media file(s), 2 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20240318_191032-ts55xd19/logs
root@608dbef3510b:~/Huatuo-Llama-Med-Chinese#
root@608dbef3510b:~/Huatuo-Llama-Med-Chinese#

```

出错日志

```

root@608dbef3510b:~/Huatuo-Llama-Med-Chinese# python finetune.py --base_model
'/home/models/llama-7b-hf' --data_path './data/llama_data.json' --
output_dir './lora-llama-med-e1' --prompt_template_name 'med_template' --
micro_batch_size 128 --batch_size 128 --wandb_run_name e1
Training Alpaca-LoRA model with params:
base_model: /home/models/llama-7b-hf
data_path: ./data/llama_data.json
output_dir: ./lora-llama-med-e1
batch_size: 128
micro_batch_size: 128
num_epochs: 10
learning_rate: 0.0003
cutoff_len: 256
val_set_size: 500
lora_r: 8
lora_alpha: 16
lora_dropout: 0.05
lora_target_modules: ['q_proj', 'v_proj']
train_on_inputs: False
group_by_length: False
wandb_project: llama_med
wandb_run_name: e1
wandb_watch:
wandb_log_model:
resume_from_checkpoint: False
prompt_template: med_template

```

```
Loading checkpoint shards: 100%|██████████|  
2/2 [00:14<00:00, 7.03s/it]  
Generating train split: 8658 examples [00:00, 344009.06 examples/s]  
trainable params: 4194304 || all params: 6742609920 || trainable%:  
0.06220594176090199  
Map: 100%|██████████|  
8158/8158 [00:06<00:00, 1320.40 examples/s]  
Map: 100%|██████████|  
500/500 [00:00<00:00, 1301.08 examples/s]  
huggingface/tokenizers: The current process just got forked, after parallelism  
has already been used. Disabling parallelism to avoid deadlocks...  
To disable this warning, you can either:  
- Avoid using `tokenizers` before the fork if possible  
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |  
false)  
huggingface/tokenizers: The current process just got forked, after parallelism  
has already been used. Disabling parallelism to avoid deadlocks...  
To disable this warning, you can either:  
- Avoid using `tokenizers` before the fork if possible  
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |  
false)  
huggingface/tokenizers: The current process just got forked, after parallelism  
has already been used. Disabling parallelism to avoid deadlocks...  
To disable this warning, you can either:  
- Avoid using `tokenizers` before the fork if possible  
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |  
false)  
huggingface/tokenizers: The current process just got forked, after parallelism  
has already been used. Disabling parallelism to avoid deadlocks...  
To disable this warning, you can either:  
- Avoid using `tokenizers` before the fork if possible  
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |  
false)  
huggingface/tokenizers: The current process just got forked, after parallelism  
has already been used. Disabling parallelism to avoid deadlocks...  
To disable this warning, you can either:  
- Avoid using `tokenizers` before the fork if possible  
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |  
false)  
huggingface/tokenizers: The current process just got forked, after parallelism  
has already been used. Disabling parallelism to avoid deadlocks...  
To disable this warning, you can either:  
- Avoid using `tokenizers` before the fork if possible  
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |  
false)  
wandb: (1) Create a W&B account  
wandb: (2) Use an existing W&B account
```

```
wandb: (3) Don't visualize my results
wandb: Enter your choice: 2
wandb: You chose 'Use an existing W&B account'
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally:
https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
wandb: Appending key for api.wandb.ai to your netrc file: /home/.netrc
wandb: Tracking run with wandb version 0.16.4
wandb: Run data is saved locally in /home/Huatuo-Llama-Med-Chinese/wandb/run-
20240318_191032-ts55xdi9
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run e1
wandb: ⭐ View project at https://wandb.ai/ember404/llama_med
wandb: 🚀 View run at https://wandb.ai/ember404/llama_med/runs/ts55xdi9
0%|
| 0/640 [00:00<?, ?it/s] You're using a
LlamaTokenizerFast tokenizer. Please note that with a fast tokenizer, using the
`__call__` method is faster than using a method to encode the text followed by a
call to the `pad` method to get a padded encoding.
/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/utils/checkpoint.py:429: UserWarning: torch.utils.checkpoint:
please pass in use_reentrant=True or use_reentrant=False explicitly. The default
value of use_reentrant will be updated to be False in the future. To maintain
current behavior, pass use_reentrant=True. It is recommended that you use
use_reentrant=False. Refer to docs for more details on the differences between
the two variants.
    warnings.warn(
/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/bitsandbytes/autograd/_functions.py:322: UserWarning: MatMul8bitLt:
inputs will be cast from torch.float32 to float16 during quantization
    warnings.warn(f"MatMul8bitLt: inputs will be cast from {A.dtype} to float16
during quantization")
Traceback (most recent call last):
  File "/home/Huatuo-Llama-Med-Chinese/finetune.py", line 289, in <module>
    fire.Fire(train)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py",
line 143, in Fire
    component_trace = _Fire(component, args, parsed_flag_args, context, name)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py",
line 477, in _Fire
    component, remaining_args = _CallAndUpdateTrace(
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py",
line 693, in _CallAndUpdateTrace
    component = fn(*varargs, **kwargs)
  File "/home/Huatuo-Llama-Med-Chinese/finetune.py", line 279, in train
    trainer.train(resume_from_checkpoint=resume_from_checkpoint)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/transformers/trainer.py", line 1662, in train
    return inner_training_loop(
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/transformers/trainer.py", line 1929, in _inner_training_loop
    tr_loss_step = self.training_step(model, inputs)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/transformers/trainer.py", line 2709, in training_step
    self.scaler.scale(loss).backward()
```

```
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/_tensor.py", line 492, in backward
    torch.autograd.backward(
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/autograd/__init__.py", line 251, in backward
    Variable._execution_engine.run_backward( # Calls into the C++ engine to run
the backward pass

    File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/autograd/function.py", line 288, in apply
        return user_fn(self, *args)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/utils/checkpoint.py", line 288, in backward
    torch.autograd.backward(outputs_with_grad, args_with_grad)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/autograd/__init__.py", line 251, in backward
    Variable._execution_engine.run_backward( # Calls into the C++ engine to run
the backward pass
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 1.34 GiB. GPU
0 has a total capacity of 47.54 GiB of which 857.00 MiB is free. Process 3124114
has 46.68 GiB memory in use. Of the allocated memory 39.27 GiB is allocated by
PyTorch, and 7.08 GiB is reserved by PyTorch but unallocated. If reserved but
unallocated memory is large try setting max_split_size_mb to avoid fragmentation.
See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
Traceback (most recent call last):
  File "/home/Huatuo-Llama-Med-Chinese/finetune.py", line 289, in <module>
    fire.Fire(train)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py",
line 143, in Fire
    component_trace = _Fire(component, args, parsed_flag_args, context, name)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py",
line 477, in _Fire
    component, remaining_args = _CallAndUpdateTrace(
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/fire/core.py",
line 693, in _CallAndUpdateTrace
    component = fn(*varargs, **kwargs)
  File "/home/Huatuo-Llama-Med-Chinese/finetune.py", line 279, in train
    trainer.train(resume_from_checkpoint=resume_from_checkpoint)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/transformers/trainer.py", line 1662, in train
    return inner_training_loop(
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/transformers/trainer.py", line 1929, in _inner_training_loop
    tr_loss_step = self.training_step(model, inputs)
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/transformers/trainer.py", line 2709, in training_step
    self.scaler.scale(loss).backward()
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/_tensor.py", line 492, in backward
    torch.autograd.backward(
  File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/autograd/__init__.py", line 251, in backward
    Variable._execution_engine.run_backward( # Calls into the C++ engine to run
the backward pass
    File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/autograd/function.py", line 288, in apply
        return user_fn(self, *args)
```

```
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/utils/checkpoint.py", line 288, in backward
    torch.autograd.backward(outputs_with_grad, args_with_grad)
File "/root/miniconda3/envs/py3.10/lib/python3.10/site-
packages/torch/autograd/__init__.py", line 251, in backward
    Variable._execution_engine.run_backward( # calls into the C++ engine to run
the backward pass
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 1.34 GiB. GPU
0 has a total capacity of 47.54 GiB of which 857.00 MiB is free. Process 3124114
has 46.68 GiB memory in use. Of the allocated memory 39.27 GiB is allocated by
PyTorch, and 7.08 GiB is reserved by PyTorch but unallocated. If reserved but
unallocated memory is large try setting max_split_size_mb to avoid fragmentation.
See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
wandb: 🔍 View run e1 at: https://wandb.ai/ember404/llama_med/runs/ts55xd19 wandb:
⚡ View job at
https://wandb.ai/ember404/llama_med/jobs/
QXJ0awZhbY3RDb2xsZWN0aw9u0jE1MDA5NDQ0NA== /version_details/v0
wandb: Synced 6 W&B file(s), 0 media file(s), 2 artifact file(s) and 0 other
file(s)
wandb: Find logs at: ./wandb/run-20240318_191032-ts55xd19/logs
```

第二次运行，调整参数

```
mkdir lora-llama-med-e2
```

```
python finetune.py \
--base_model '/home/models/llama-7b-hf' \
--data_path './data/llama_data.json' \
--output_dir './lora-llama-med-e2' \
--prompt_template_name 'med_template' \
--micro_batch_size 64 \
--batch_size 64 \
--wandb_run_name e2
```

正常运行

```
Terminal 5      Terminal 6      finetune.py      finetune.sh      infer.sh
root@608dbef3510b:/Huatuo-Llama-Med-Chinese# python finetune.py \
--base_model '/home/models/llama-7b-hf' \
--data_path './data/llama_data.json' \
--output_dir './lora-llama-med-e2' \
--prompt_template_name 'med_template' \
--micro_batch_size 64 \
--batch_size 64 \
--wandb_run_name e2
Training Alpaca-LoRA model with params:
base_model: /home/models/llama-7b-hf
data_path: ./data/llama_data.json
output_dir: ./lora-llama-med-e2
batch_size: 64
micro_batch_size: 64
num_epochs: 10
learning_rate: 0.0003
cutoff_len: 256
val_set_size: 500
lora_r: 8
lora_alpha: 16
lora_dropout: 0.05
lora_target_modules: ['q_proj', 'v_proj']
train_on_inputs: False
group_by_length: False
wandb_project: llama_med
wandb_run_name: e2
wandb_watch:
wandb_log_model:
resume_from_checkpoint: False
prompt template: med_template

Loading checkpoint shards: 100% |██████████| 2/2 [00:13<00:00,  6.91s/it]
trainable params: 4194304 || all params: 6742609920 || trainable%: 0.06220594176090199
Map: 100% |██████████| 8158/8158 [00:06<00:00, 1247.00 examples/s]
Map: 100% |██████████| 500/500 [00:00<00:00, 1304.16 examples/s]
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
```

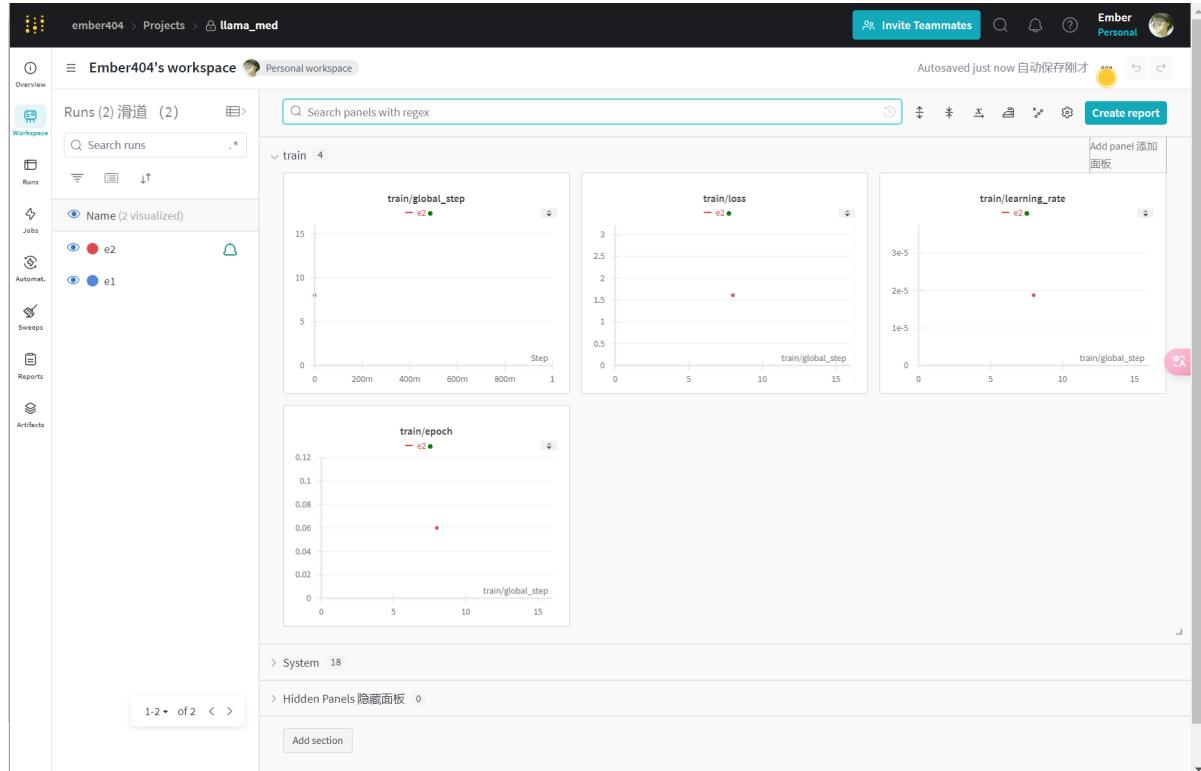
```

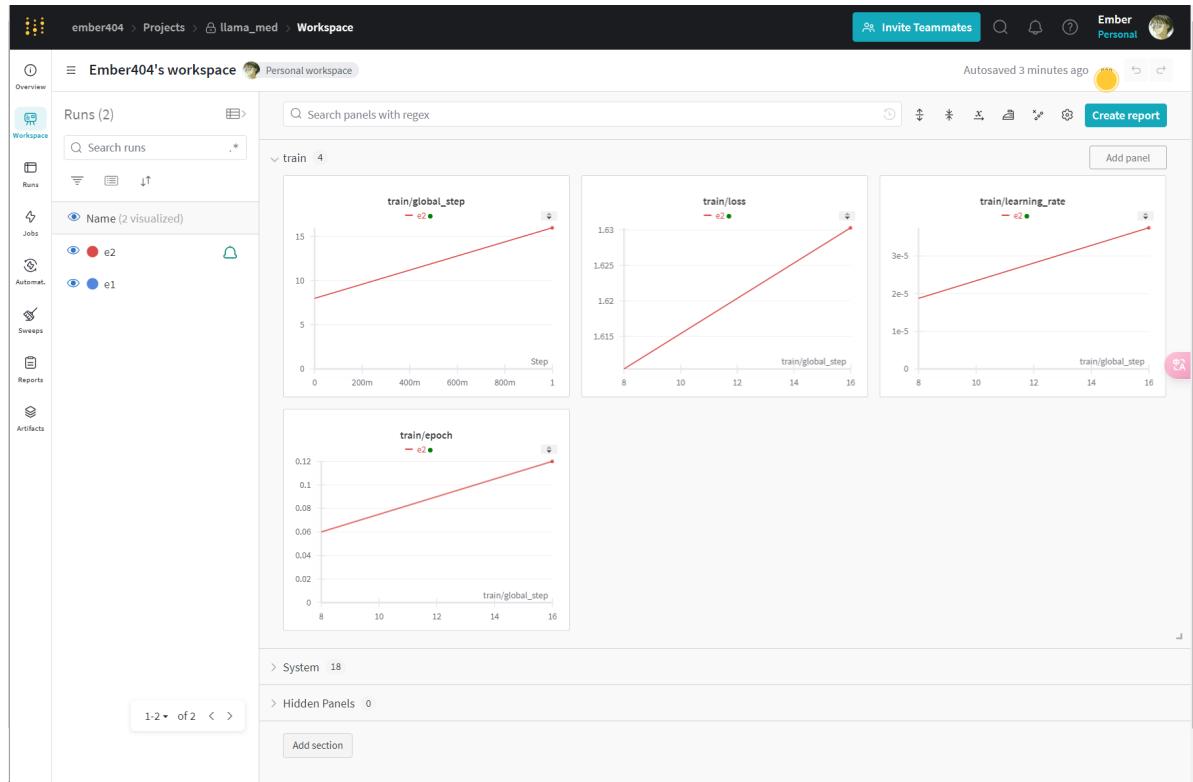
Loading checkpoint shards: 100% | 2/2 [00:13<00:00, 6.91s/it]
trainable params: 4194304 || all params: 6742609920 || trainable%: 0.06220594176090199
Map: 100% | 8158/8158 [00:06<00:00, 1247.00 examples/s]
Map: 100% | 500/500 [00:00<00:00, 1304.16 examples/s]
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
wandb: Currently logged in as: ember404. Use wandb login --relogin to force relogin
wandb: Tracking run with wandb version 0.16.4
wandb: Run data is saved locally in /home/Huatuo-Llama-Med-Chinese/wandb/run-20240318_192200-dnug9898
wandb: Run wandb offline to turn off syncing.
wandb: Syncing run e2
wandb: ★View project at https://wandb.ai/ember404/llama\_med
wandb: 🌟View run at https://wandb.ai/ember404/llama\_med/runs/dnug9898
0% | 0/1280 [00:00<?, ?it/s] You're using LlamaTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using a method to encode the text followed by a call to the `pad` method to get a padded encoding.
/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/utils/checkpoint.py:429: UserWarning: torch.utils.checkpoint: please pass in use_reentrant=True or use_reentrant=False explicitly. The default value of use_reentrant will be updated to be False in the future. To maintain current behavior, use_reentrant=True. It is recommended that you use use_reentrant=False. Refer to docs for more details on the differences between the two variants.
warnings.warn(
/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/bitsandbytes/autograd/_functions.py:322: UserWarning: MatMul8bitLt: inputs will be cast from float32 to float16 during quantization
warnings.warn(f"MatMul8bitLt: inputs will be cast from {fa.dtype} to float16 during quantization")
['loss': 1.6104, 'learning_rate': 1.875e-05, 'epoch': 0.06]
['loss': 1.6303, 'learning_rate': 3.75e-05, 'epoch': 0.12]
1% | 17/1280 [05:36<6:57:25, 19.83s/it]

```

看到上面这个输出证明开始微调了

查看我的wandb





继续微调中，第7个epoch

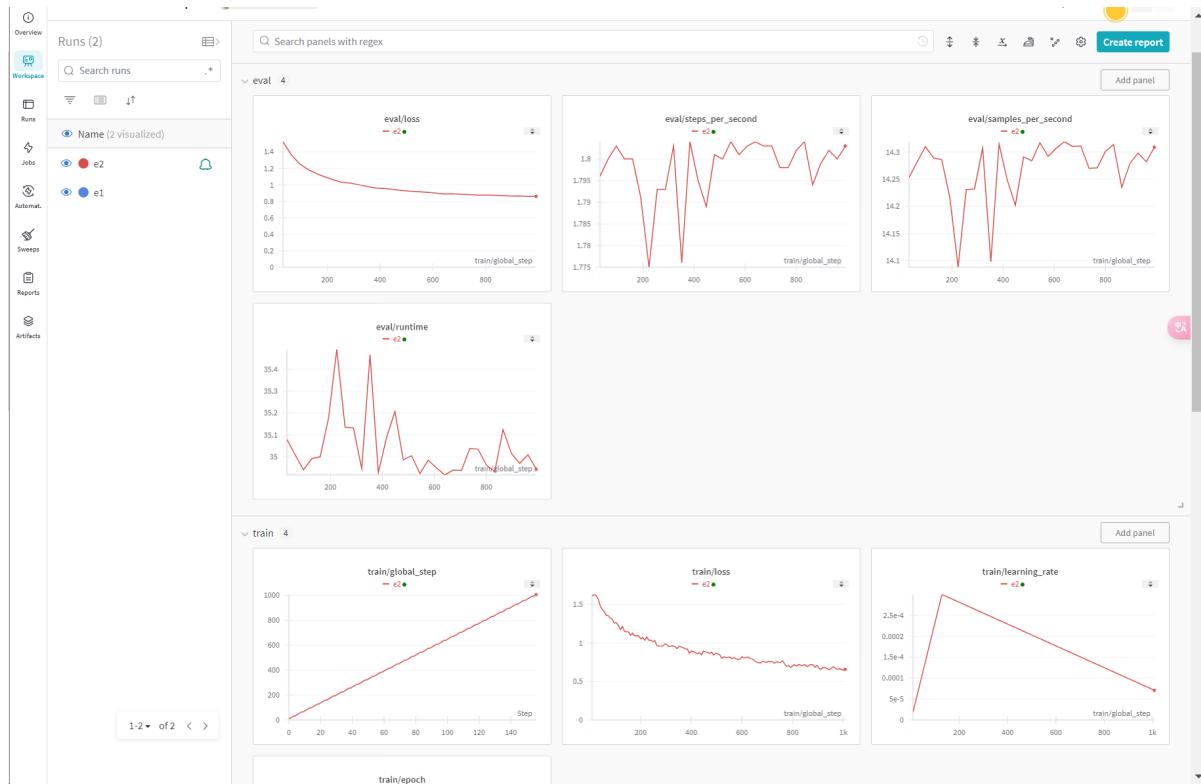
The screenshot shows a Jupyter Notebook interface. On the left, there's a file browser with a list of checkpoints. The list includes:

- checkpoint-864 (51 minutes ago)
- checkpoint-896 (40 minutes ago)
- checkpoint-928 (29 minutes ago)
- checkpoint-960 (17 minutes ago)
- checkpoint-992 (6 minutes ago)

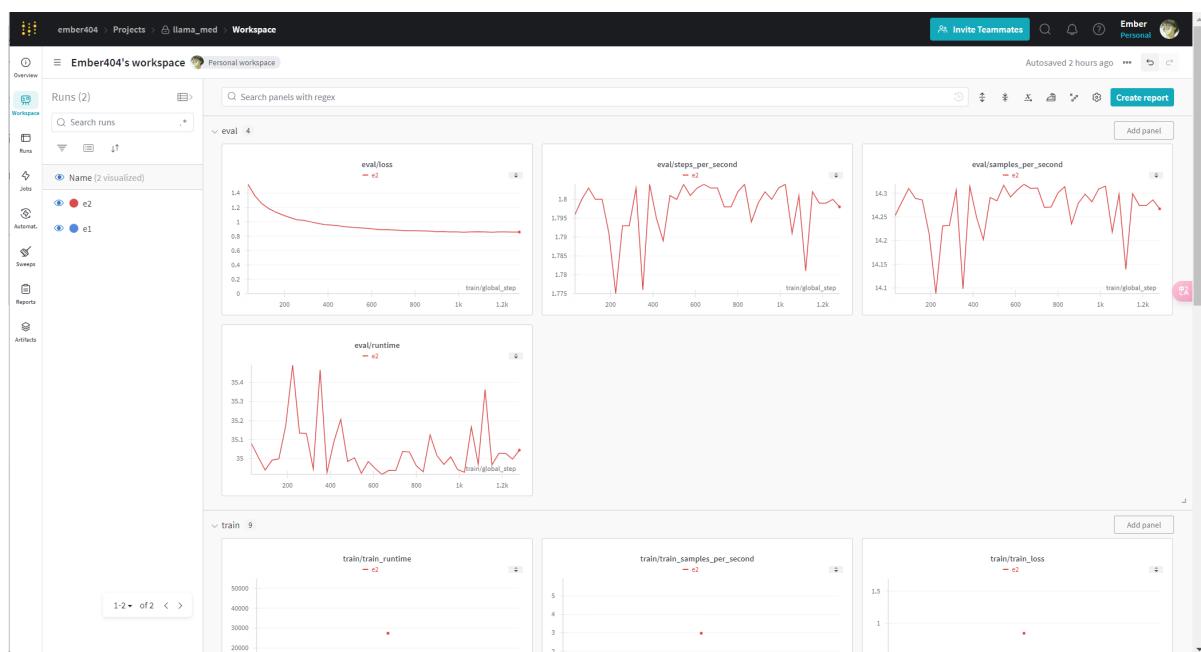
The main area has several terminal tabs open, labeled Terminal 5 through Terminal 10. The logs from these terminals show training progress for epochs 7 to 16. The logs include:

- Epoch 7: Loss 0.6719, Learning Rate 7.06e-05
- Epoch 8: Loss 0.6579, Learning Rate 7.12e-05
- Epoch 9: Loss 0.682, Learning Rate 7.19e-05
- Epoch 10: Loss 0.679, Learning Rate 7.25e-05
- Epoch 11: Loss 0.8665027618405203, Eval Runtime 34.9703, Eval Samples Per Second 14.295, Epoch 7.25
- Epoch 12: Loss 0.6719, Learning Rate 7.31e-05
- Epoch 13: Loss 0.6499, Learning Rate 7.38e-05
- Epoch 14: Loss 0.6706, Learning Rate 7.44e-05
- Epoch 15: Loss 0.6905, Learning Rate 7.5e-05
- Epoch 16: Loss 0.8611739277539061, Eval Runtime 35.0097, Eval Samples Per Second 14.282, Epoch 7.5
- Epoch 17: Loss 0.6762, Learning Rate 7.56e-05
- Epoch 18: Loss 0.657, Learning Rate 7.62e-05
- Epoch 19: Loss 0.666, Learning Rate 7.69e-05
- Epoch 20: Loss 0.6576, Learning Rate 7.75e-05
- Epoch 21: Loss 0.860729457826343, Eval Runtime 34.9454, Eval Samples Per Second 14.309, Epoch 7.8
- Epoch 22: Loss 0.6762, Learning Rate 7.86e-05
- Epoch 23: Loss 0.657, Learning Rate 7.92e-05
- Epoch 24: Loss 0.666, Learning Rate 7.98e-05
- Epoch 25: Loss 0.6572, Learning Rate 7.88e-05
- Epoch 26: Loss 0.8610729457826343, Eval Runtime 1010.1280, Epoch 8.0

当前可视化



最终可视化



微调完成，10个epoch，运行了7个小时

File Edit View Run Kernel Tabs Settings Help

Terminal 5 Terminal 6 finetune.py finetune.sh infer.sh

Filter files by name

/ Huatu-LLama-Med-Chinese / lora-llama-med-e2/

Name	Last Modified
checkpoint-1152	3 hours ago
checkpoint-1184	3 hours ago
checkpoint-1216	3 hours ago
checkpoint-1248	3 hours ago
checkpoint-1280	3 hours ago
adapter config.json	3 hours ago
adapter model.bin	3 hours ago

[eval loss': 0.858422756190684, 'eval runtime': 35.0451, 'eval samples per second': 14.267, 'eval steps_per_second': 1.798, 'epoch': 10.0]
100% 1280/1280 [77:36:55/00:00, 16.73s/it]
100%

Could not locate the best model at ./lora-llama-med-e2/checkpoint-1152/pytorch_model.bin, if you are running a distributed training on multiple nodes, you should activate --save_on_each_node.

{'train_runtime': 27455.7476, 'train samples per second': 2.973, 'train steps_per_second': 0.047, 'train_loss': 0.846734871715307, 'epoch': 10.0}
100% 1280/1280 [57:31:12/00:00, 21.43s/it]

If there's a warning about missing keys above, please disregard :)

wandb: = 0.147 MB of 0.147 MB uploaded (0.004 MB deduced)

wandb: Run history:

wandb: eval loss eval.runtime eval.samples_per_second eval.steps_per_second eval.step.train_epoch train.global_step train.learning_rate train.loss train.total_flos train.train.loss train.train.losses train.train.step train.train_samples_per_second train.train.steps_per_second

wandb: Run summary:

wandb: eval.loss 0.85842 eval.runtime 35.0451 eval.samples_per_second 14.267 eval.steps_per_second 1.798 train.epoch 10.0 train.global_step 1280 train.learning_rate 0.0 train.loss 0.846734871715307 train.total_flos 8.294711887568899e+17 train.train.loss 0.84673 train.train.runtime 27455.7476 train.train_samples_per_second 2.973 train.train.steps_per_second 0.047

wandb: #View job at https://wandb.ai/hantu-llama-ml/lora-llama-med-e2/run/mn8989
wandb: #View job at https://wandb.ai/hantu-llama-ml/lora-llama-med-e2/run/mn8989?cv=2&N=990#M1DA5DQVA=/version/details/v1
wandb: Synced 6 WB file(s), 0 media file(s), 2 artifact file(s) and 0 other file(s)
wandb: Find loss at ./wandb/run/20240318_192200-drag8989/logs
root@605b0ef3510b:/Huatu-LLama-Med-Chinese\$

结束日志

```
{'train_runtime': 27435.7476, 'train_samples_per_second': 2.973,  
'train_steps_per_second': 0.047, 'train_loss': 0.8467348717153073, 'epoch': 10.0}  
100%|
```

[1280/1280 [7:37:12<00:00, 21.43s/it]

If there's a warning

wandb: - 0.147 MB of 0.147 MB uploaded

wandb: Run history:

Wahab:

wandb eval /nuptime

Wandb: eval/runtime

wandb: eval/samples per second

Figure 1. The distribution of the number of species per genus in the *Acacia* alliance.

wandb: eval/steps_per_second

A horizontal bar chart with 20 bars. The bars are dark grey and of varying heights. The distribution is highly skewed, with most bars being very short and a few being very tall.

wandb: train/epoch

wandb: train/global_step

wandb: train/learning_rate

—
—

wandb: train/loss

wandb: train/total_floss

wandb: train/total_f1s
wandb: train/train_loss

wandb: train/train_runtime

wandb: train/train samples per second

wandb: train/train_steps_per_second

wandb:

wandb: Run summary:

wandb: eval/loss

wandb: eval/runtime

wandb: eval/samples_per_second

```
wandb: eval/steps_per_second 1.798
wandb: train/epoch 10.0
wandb: train/global_step 1280
wandb: train/learning_rate 0.0
wandb: train/loss 0.5994
wandb: train/total_flos 8.284711887568896e+17
wandb: train/train_loss 0.84673
wandb: train/train_runtime 27435.7476
wandb: train/train_samples_per_second 2.973
wandb: train/train_steps_per_second 0.047
wandb:
wandb: 🚀 view run e2 at: https://wandb.ai/ember404/llama_med/runs/dnug9898
wandb: ⚡ View job at
https://wandb.ai/ember404/llama_med/jobs/QXJ0awZhY3RDb2xsZWN0aw9u0jE1MDA5NDQ0NA==/
version_details/v1
wandb: Synced 6 W&B file(s), 0 media file(s), 2 artifact file(s) and 0 other
file(s)
wandb: Find logs at: ./wandb/run-20240318_192200-dnug9898/logs
```

可视化查看

https://wandb.ai/ember404/llama_med/runs/dnug9898?nw=nwuserember404

