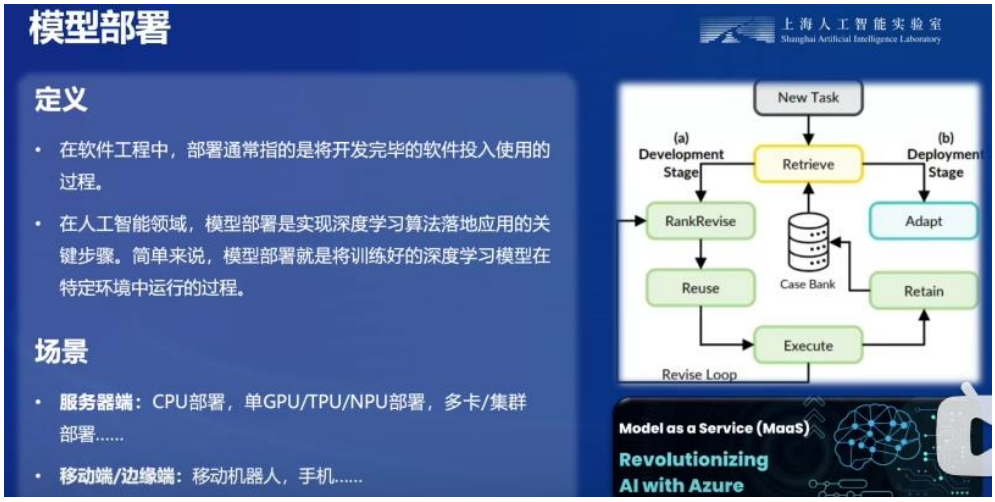
Lesson 5 笔记

LMDeploy 量化部署 LLM-VLM 实践

这节课分为 4 个部分大模型部署背景、方法、LMDeployi 简介和动手实践环节。



访存瓶颈

- 大模型推理是“访存密集”型任务。目前硬件计算速度“远快于”显存带宽，存在严重的访存性能瓶颈。
- 以RTX 4090推理175B大模型为例，BS为1时计算量为6.83TFLOPs，远低于82.58 TFLOPs的FP16计算能力；但访存量为32.62 TB，是显存带宽每秒处理能力的30倍。

动态请求

- 请求量不确定；
- 请求时间不确定；
- Token逐个生成，生成数量不确定。

GPT3-175B推理阶段计算访存比分析 (输入1k，输出250) [1]:

BS	计算量	访存量	计算访存比
1	6.83 TFLOPs	32.62 TB	0.20
8	55.37 TFLOPs	32.67 TB	1.67
16	112.3 TFLOPs	32.73 TB	3.43

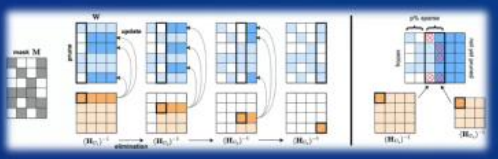
常见GPU浮点运算性能与内存带宽[2]:

GPU	FP16算力	FP32算力	FP64算力	显存带宽	FP16算力/显存带宽
RTX 4090	82.58 TFLOPs	82.58 TFLOPs	1290 GFLOPs	1008 GB/s	81.92
A100 80G	77.97 TFLOPs	19.49 TFLOPs	9.746 TFLOPs	2039 GB/s	38.24
H100 80G	267.6 TFLOPs	66.91 TFLOPs	33.45 TFLOPs	1681 GB/s	159.2

模型剪枝(Pruning)

上海人工智能实验室 Shanghai Artificial Intelligence Laboratory

剪枝指移除模型中不必要或多余的组件，比如参数，以使模型更加高效。通过对模型中贡献有限的冗余参数进行剪枝，在保证性能最低下降的同时，可以减小存储需求、提高计算效率。



非结构化剪枝 SparseGPT[1]，LoRAPrune[2]，Wanda[3]

- 指移除个别参数，而不考虑整体网络结构。这种方法通过将低于阈值的参数置零的方式对个别权重或神经元进行处理。

结构化剪枝 LLM-Pruner[4]

- 根据预定义规则移除连接或分层结构，同时保持整体网络结构。这种方法一次性地针对整组权重，优势在于降低模型复杂性和内存使用，同时保持整体的LLM结构完整。

Reference:

[1] Frantar E, Alistarh D. Sparsegpt: Massive language models can be accurately pruned in one-shot[C]//International Conference on Machine Learning. PMLR, 2023: 10323-10337.

[2] Zhang M, Chen H, Shen C, et al. Loraprune: Pruning meets low-rank parameter-efficient fine-tuning[J]. 2023.

[3] Sun M, Liu Z, Bair A, et al. A simple and effective pruning approach for large language models[J]. arXiv preprint arXiv:2306.11695, 2023.

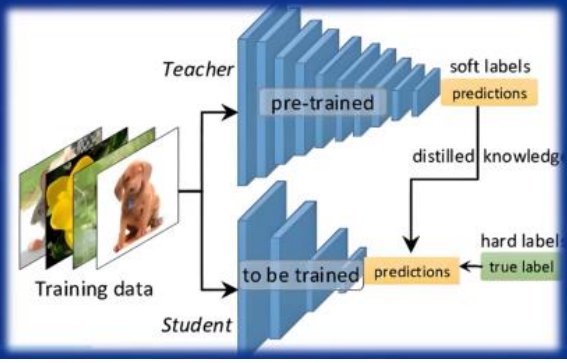
[4] Ma X, Fang G, Wang X. Llm-pruner: On the structural pruning of large language models[J]. Advances in neural information processing systems, 2023, 36: 21702-21720.

知识蒸馏(Knowledge Distillation，KD)

上海人工智能实验室 Shanghai Artificial Intelligence Laboratory

知识蒸馏是一种经典的模型压缩方法，核心思想是通过引导轻量化的学生模型“模仿”性能更好、结构更复杂的教师模型，在不改变学生模型结构的情况下提高其性能。

- **上下文学习(ICL)**：ICL distillation[1]
- **思维链(CoT)**：MT-COT[2]，Fine-tune-CoT[3]等
- **指令跟随(IF)**：LaMini-LM[4]



Reference:

[1] Wu M, Waheed A, Zhang C, et al. Lamini-lm: A diverse herd of distilled models from large-scale instructions[J]. arXiv preprint arXiv:2304.14402, 2023.

[2] Li S, Chen J, Shen Y, et al. Explanations from large language models make small reasoners better[J]. arXiv preprint arXiv:2210.06726, 2022.

[3] Ho N, Schmid L, Yun S Y. Large language models are reasoning teachers[J]. arXiv preprint arXiv:2212.10071, 2022.

[4] Huang Y, Chen Y, Yu Z, et al. In-context learning distillation: Transferring few-shot learning ability of pre-trained language models[J]. arXiv preprint arXiv:2212.10670, 2022.

量化(Quantization)

上海人工智能实验室 Shanghai Artificial Intelligence Laboratory

量化技术将传统的表示方法中的浮点数转换为整数或其他离散形式，以减轻深度学习模型的存储和计算负担。

量化感知训练(QAT) LLM-QAT[1]

- 量化目标无缝地集成到模型的训练过程中。这种方法使LLM在训练过程中适应低精度表示.

量化感知微调(QAF) PEQA[2]，QLORA[3]

- QAF涉及在微调过程中对LLM进行量化。主要目标是确保经过微调的LLM在量化为较低位宽后仍保持性能。

训练后量化(PTQ) LLM.int8[4]，AWQ[5]

- 在LLM的训练阶段完成后对其参数进行量化。PTQ的主要目标是减少LLM的存储和计算复杂性，而**无需对LLM架构进行修改或进行重新训练。**

通用公式：

$$ZP = \frac{\min + \max}{2} \qquad S = \frac{\max - \min}{255}$$

量 化：$q = \text{round}\left(\frac{f - ZP}{S}\right)$

反量化：$f = q \times S + ZP$

Reference：

[1] Liu Z, Oguz B, Zhao C, et al. Llm-qat: Data-free quantization aware training for large language models[J]. arXiv preprint arXiv:2305.17888, 2023.

[2] Arshia F Z, Keyvanrad M A, Sadidpour S S, et al. PeQA: A Massive Persian Question-Answering and Chatbot Dataset[C]//2022 12th International Conference on Computer and Knowledge Engineering (ICCKE). IEEE, 2022: 392-397.

[3] Dettmers T, Pagnoni A, Holtzman A, et al. Qlora: Efficient finetuning of quantized llms[J]. Advances in Neural Information Processing Systems, 2024, 36.

[4] Dettmers T, Lewis M, Belkada Y, et al. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale[J]. Advances in Neural Information Processing Systems, 2022, 35: 30318-30332.

[5] Lin J, Tang J, Tang H, et al. Awq: Activation-aware weight quantization for llm compression and acceleration[J]. arXiv preprint arXiv:2306.00978, 2023.

LMDeploy简介

上海人工智能实验室 Shanghai Artificial Intelligence Laboratory

LMDeploy 由 MMDeploy 和 MMRazor 团队联合开发，是涵盖了 LLM 任务的全套轻量化、部署和服务解决方案。核心功能包括高效推理、可靠量化、便捷服务和有状态推理。

LMDeploy

- **高效的推理：**LMDeploy开发了Continuous Batch，Blocked K/V Cache，动态拆分和融合，张量并行，高效的计算kernel等重要特性。InternLM2推理性能是vLLM的 1.8 倍。
- **可靠的量化：**LMDeploy支持权重量化和k/v量化。4bit模型推理效率是FP16下的2.4倍。量化模型的可靠性已通过OpenCompass评测得到充分验证。
- **便捷的服务：**通过请求分发服务，LMDeploy 支持多模型在多机、多卡上的推理服务。
- **有状态推理：**通过缓存多轮对话过程中Attention的k/v，记住对话历史，从而避免重复处理历史会话。显著提升长文本多轮对话场景中的效率。

LMDeploy核心功能

上海人工智能实验室 Shanghai Artificial Intelligence Laboratory

模型高效推理 参考命令：lmdeploy chat -h

- TurboMind是LMDeploy团队开发的一款关于 LLM 推理的高效推理引擎。它的主要功能包括：LLaMa 结构模型的支持，continuous batch推理模式和可扩展的 KV 缓存管理器。

模型量化压缩 参考命令：lmdeploy lite -h

- **W4A16量化(AWQ)：**将 FP16 的模型权重量化为 INT4，Kernel 计算时，访存量直接降为 FP16 模型的 1/4，大幅降低了访存成本。Weight Only 是指仅量化权重，数值计算依然采用 FP16（需要将 INT4 权重反量化）。

服务化部署 参考命令：lmdeploy serve -h

- 将LLM封装为HTTP API服务，支持Triton扩展。

新版本的 LMDeploy(推理视觉多模态大模型)支持了对多模态大模型 llava 的支持！可以使用 pipeline 便捷运行。

模型	参数量	模型	参数量
Llama	7B-65B	Llama2	7B-70B
InternLM	7B-20B	InternLM2	1.8B-20B
Llava	7B-13B	InternLM-XComposer	7B
QWen	7B-72B	Qwen-VL	7B
QWen1.5	0.5B-72B	QWen1.5-MoE	A2.7B
Baichuan	7B-13B	Baichuan2	7B-13B
Code Llama	7B-34B	ChatGLM2	6B
Falcon	7B-180B	YI	6B-34B
Mistral	7B	Mixtral	8x7B
DeepSeek-MoE	16B	DeepSeek-VL	7B
Gemma	2B-7B	Dbrx	132B

实践记录

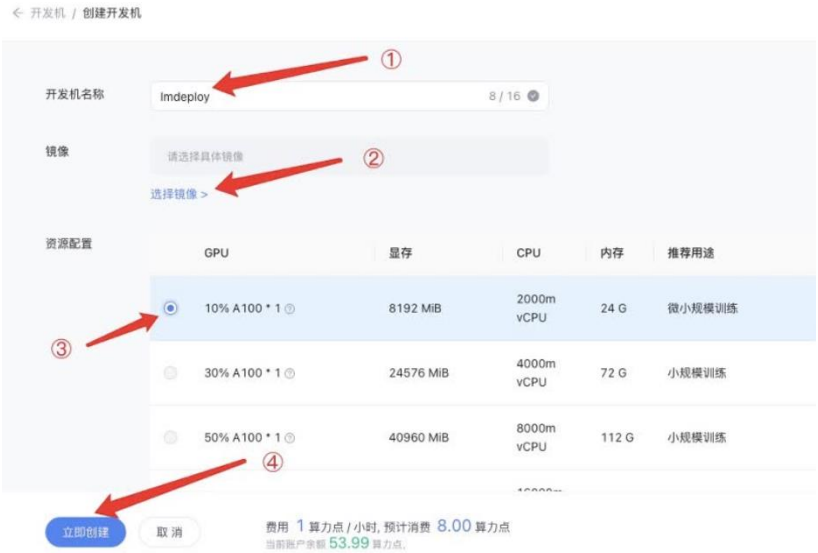
1. LMDeploy 环境部署

1.1 创建开发机

打开 InternStudio 平台，创建开发机。

填写开发机名称；选择镜像 Cuda12.2-conda；选择 10% A100*1GPU；点击“立即创建”。注意请不要选择 Cuda11.7-conda 的镜像，新版本的 lmdeploy 会出现兼容性问

题。



然后进入开发机，切换为终端(Terminal)模式。

1.2 创建 conda 环境

InternStudio 开发机创建 conda 环境（推荐）

由于环境依赖项存在 torch，下载过程可能比较缓慢。InternStudio 上提供了快速创建 conda 环境的方法。打开命令行终端，创建一个名为 lmdeploy 的环境：

studio-conda -t lmdeploy -o pytorch-2.1.2

1.3 安装 LMDeploy

conda activate lmdeploy

pip install lmdeploy[all]==0.3.0

2. LMDeploy 模型对话(chat)

2.1 Huggingface 与 TurboMind

托管在 HuggingFace 社区的模型通常采用 HuggingFace 格式存储，简写为 HF 格式。

TurboMind 是 LMDeploy 团队开发的一款关于 LLM 推理的高效推理引擎，它的主要功能包括：LLaMa 结构模型的支持，continuous batch 推理模式和可扩展的 KV 缓存管理器。

TurboMind 推理引擎仅支持推理 TurboMind 格式的模型。因此，TurboMind 在推理 HF 格式的模型时，会首先自动将 HF 格式模型转换为 TurboMind 格式的模型。该过程在新版本的 LMDeploy 中是自动进行的，无需用户操作。

几个容易迷惑的点：

- TurboMind 与 LMDeploy 的关系：LMDeploy 是涵盖了 LLM 任务全套轻量化、部署和服务解决方案的集成功能包，TurboMind 是 LMDeploy 的一个推理引擎，是一个子模块。LMDeploy 也可以使用 pytorch 作为推理引擎。
- TurboMind 与 TurboMind 模型的关系：TurboMind 是推理引擎的名字，TurboMind 模型是一种模型存储格式，TurboMind 引擎只能推理 TurboMind 格式的模型。

2.2 下载模型

共享目录中准备好了常用的预训练模型，可以运行如下命令查看：

ls /root/share/new_models/Shanghai_AI_Laboratory/

以 InternLM2-Chat-1.8B 模型为例，从官方仓库下载模型。

InternStudio 开发机上下载模型（推荐）

如果你是在 InternStudio 开发机上，可以按照如下步骤快速下载模型。

首先进入一个你想要存放模型的目录，本教程统一放置在 Home 目录。执行如下指令：cd ~

然后执行如下指令由开发机的共享目录软链接或拷贝模型：

ln -s /root/share/new_models/Shanghai_AI_Laboratory/internlm2-chat-1_8b /root/

cp -r /root/share/new_models/Shanghai_AI_Laboratory/internlm2-chat-1_8b /root/

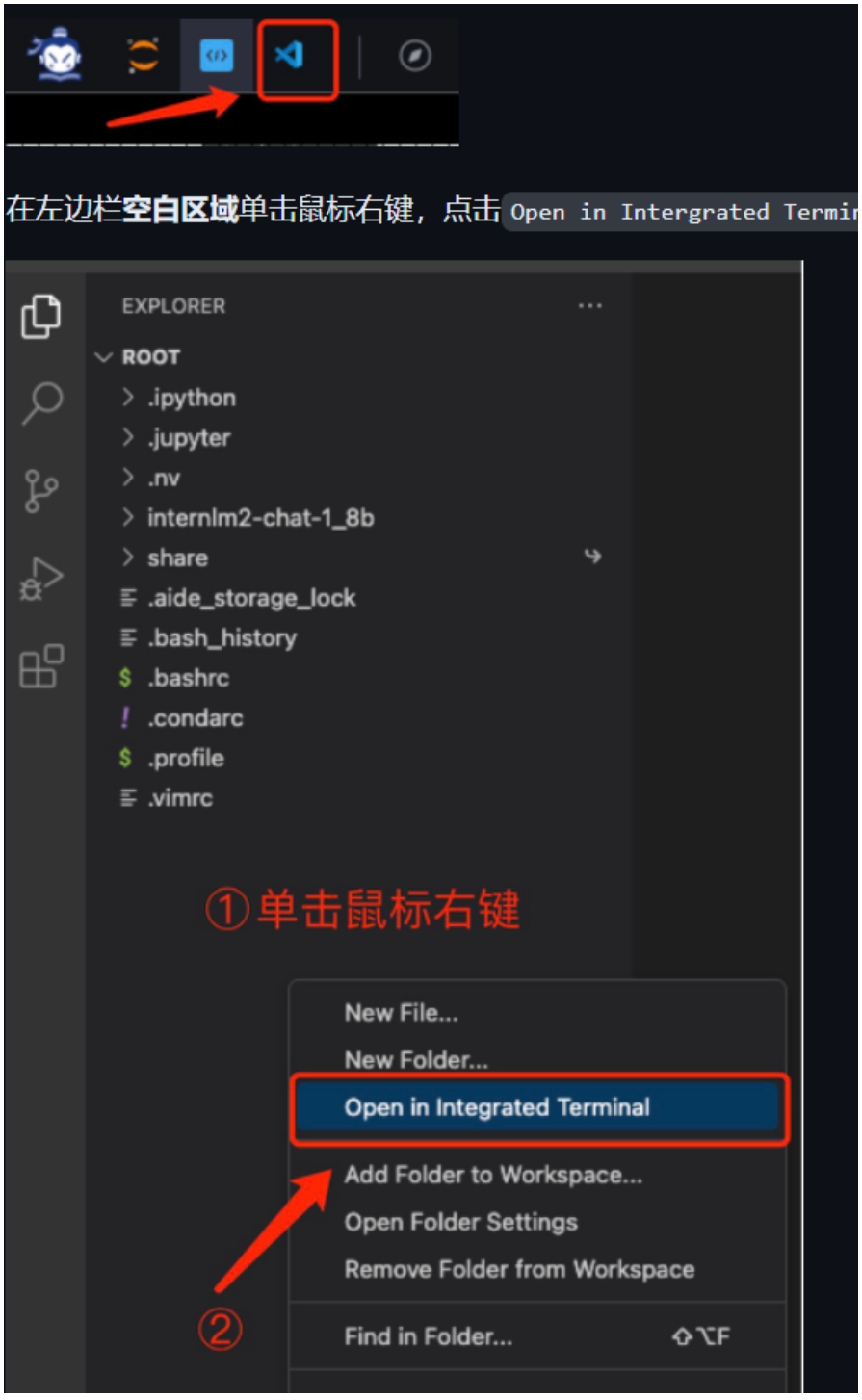
执行完如上指令后，可以运行“ls”命令。可以看到，当前目录下已经多了一个 internlm2-chat-1_8b 文件夹，即下载好的预训练模型。

2.3 使用 Transformer 库运行模型

Transformer 库是 Huggingface 社区推出的用于运行 HF 模型的官方库。

在 2.2 中，我们已经下载好了 InternLM2-Chat-1.8B 的 HF 模型。下面我们先用 Transformer 来直接运行 InternLM2-Chat-1.8B 模型，后面对比一下 LMDeploy 的使用感受。

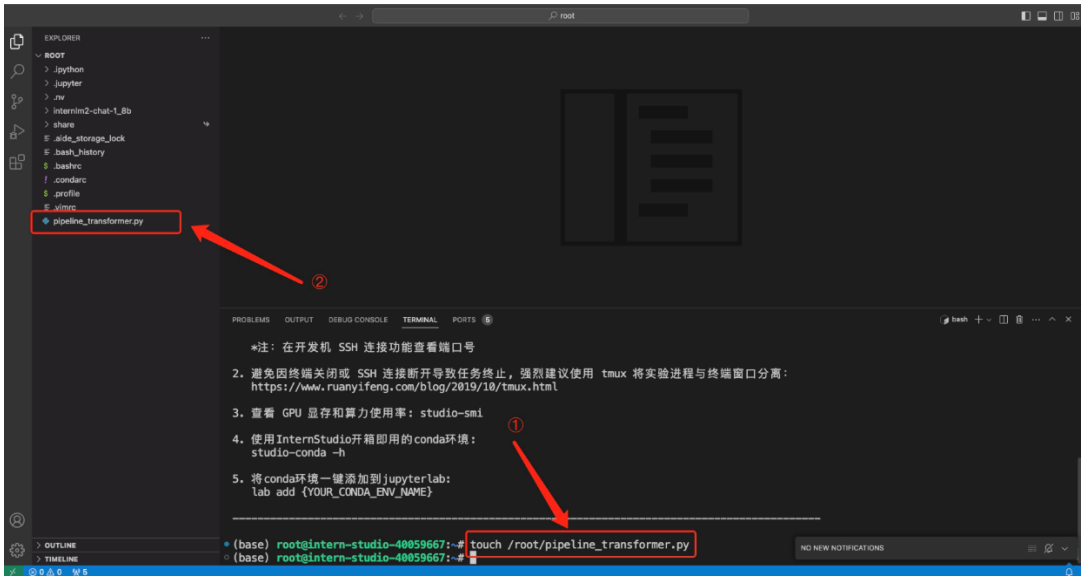
点击左上角的图标，打开 VSCode。



等待片刻，打开终端。输入如下指令，新建 pipeline_transformer.py

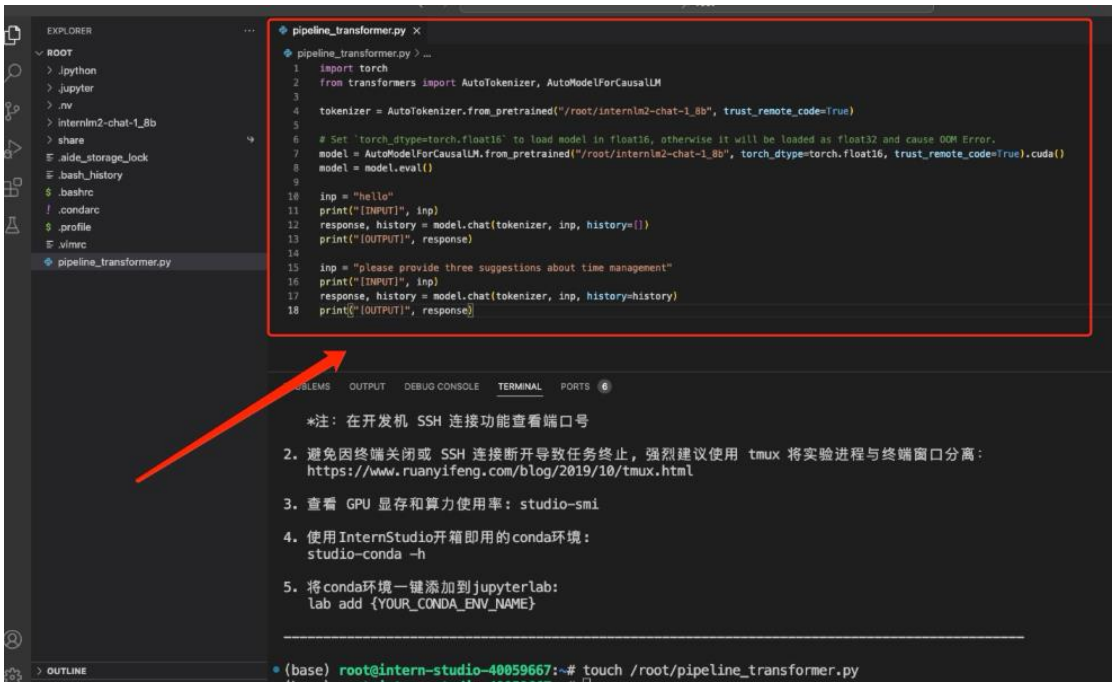
touch /root/pipeline_transformer.py

回车执行指令，可以看到侧边栏多出了 pipeline_transformer.py 文件，点击打开。后文中如果要创建其他新文件，也是采取类似的操作。



将以下内容复制粘贴进入 pipeline_transformer.py

```
import torch

from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained("/root/internlm2-chat-1_8b", trust_remote_code=True)

# Set `torch_dtype=torch.float16` to load model in float16, otherwise it will be loaded as float32 and cause OOM Error.

model = AutoModelForCausalLM.from_pretrained("/root/internlm2-chat-1_8b", torch_dtype=torch.float16, trust_remote_code=True).cuda()

model = model.eval()

inp = "hello"

print("[INPUT]", inp)

response, history = model.chat(tokenizer, inp, history=[])

print("[OUTPUT]", response)

inp = "please provide three suggestions about time management"

print("[INPUT]", inp)

response, history = model.chat(tokenizer, inp, history=history)

print("[OUTPUT]", response)
```



按 Ctrl+S 键保存（Mac 用户按 Command+S）。

回到终端，激活 conda 环境，conda activate lmdeploy

python /root/pipeline_transformer.py

2.4 使用 LMDeploy 与模型对话（应用 LMDeploy 直接与模型进行对话）

conda activate lmdeploy

使用 LMDeploy 与模型进行对话的通用命令格式为：

lmdeploy chat [HF 格式模型路径/TurboMind 格式模型路径]

执行如下命令运行下载的 1.8B 模型：

lmdeploy chat /root/internlm2-chat-1_8b

下面我们就可以与 InternLM2-Chat-1.8B 大模型对话了。比如输入“请给我讲一个小故

事吧”，然后按两下回车键。

最后输入“exit”并按两下回车，可以退出对话。

拓展内容：有关 LMDeploy 的 chat 功能的更多参数可通过-h 命令查看。

lmdeploy chat -h

3. LMDeploy 模型量化(lite)

本部分内容主要介绍如何对模型进行量化。主要包括 KV8 量化和 W4A16 量化。总的来说，量化是一种以参数或计算中间结果精度下降换空间节省（以及同时带来的性能提升）的策略。

正式介绍 LMDeploy 量化方案前，需要先介绍两个概念：

- 计算密集（compute-bound）：指推理过程中，绝大部分时间消耗在数值计算上；针对计算密集型场景，可以通过使用更快的硬件计算单元来提升计算速。
- 访存密集（memory-bound）：指推理过程中，绝大部分时间消耗在数据读取上；针对访存密集型场景，一般通过减少访存次数、提高计算访存比或降低访存量来优化。

常见的 LLM 模型由于 Decoder Only 架构的特性，实际推理时大多数的时间都消耗在了逐 Token 生成阶段（Decoding 阶段），是典型的访存密集型场景。

那么，如何优化 LLM 模型推理中的访存密集问题呢？ 我们可以使用 KV8 量化和 W4A16 量化。KV8 量化是指将逐 Token（Decoding）生成过程中的上下文 K 和 V 中间结果进行 INT8 量化（计算时再反量化），以降低生成过程中的显存占用。W4A16 量化，将 FP16 的模型权重量化为 INT4，Kernel 计算时，访存量直接降为 FP16 模型的 1/4，大幅降低了访存成本。Weight Only 是指仅量化权重，数值计算依然采用 FP16（需要将 INT4 权重反量化）。

3.1 设置最大 KV Cache 缓存大小

KV Cache 是一种缓存技术，通过存储键值对的形式来复用计算结果，以达到提高性能和降低内存消耗的目的。在大规模训练和推理中，KV Cache 可以显著减少重复计算量，从而提升模型的推理速度。理想情况下，KV Cache 全部存储于显存，以加快访存速度。当显存空间不足时，也可以将 KV Cache 放在内存，通过缓存管理器控制将当前需要使用的数据放入显存。

模型在运行时，占用的显存可大致分为三部分：模型参数本身占用的显存、KV Cache 占用的显存，以及中间运算结果占用的显存。LMDeploy 的 KV Cache 管理器可以通过设置--cache-max-entry-count 参数，控制 KV 缓存占用剩余显存的最大比例。默认的

比例为 0.8。

下面通过几个例子，看一下调整--cache-max-entry-count 参数的效果。首先保持不加该参数（默认 0.8），运行 1.8B 模型。

lmdeploy chat /root/internlm2-chat-1_8b

3.2 使用 W4A16 量化

LMDeploy 使用 AWQ 算法，实现模型 4bit 权重量化。推理引擎 TurboMind 提供了非常高效的 4bit 推理 cuda kernel，性能是 FP16 的 2.4 倍以上。它支持以下 NVIDIA 显卡：

图灵架构（sm75）：20 系列、T4

安培架构（sm80,sm86）：30 系列、A10、A16、A30、A100

Ada Lovelace 架构（sm90）：40 系列

pip install einops==0.7.0

仅需执行一条命令，就可以完成模型量化工作。

lmdeploy lite auto_awq \

```
 /root/internlm2-chat-1_8b \
--calib-dataset 'ptb' \
--calib-samples 128 \
--calib-seqlen 1024 \
--w-bits 4 \
--w-group-size 128 \
--work-dir /root/internlm2-chat-1_8b-4bit
```

量化工作结束后，新的 HF 模型被保存到 internlm2-chat-1_8b-4bit 目录。下面使用 Chat 功能运行 W4A16 量化后的模型。

```
lmdeploy chat /root/internlm2-chat-1_8b-4bit --model-format awq
```

为了更加明显体会到 W4A16 的作用，我们将 KV Cache 比例再次调为 0.01，查看显存占用情况。

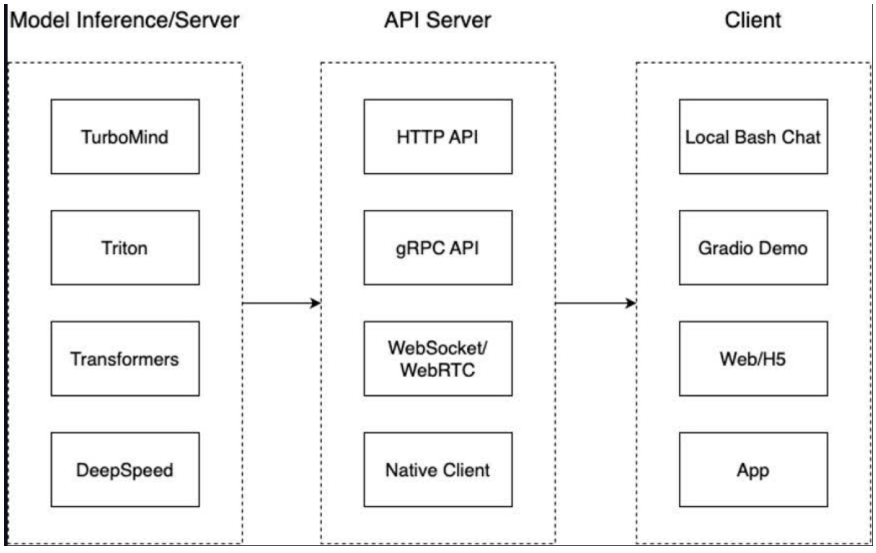
```
lmdeploy chat /root/internlm2-chat-1_8b-4bit --model-format awq --cache-max-entry-count 0.01
```

拓展内容：有关 LMDeploy 的 lite 功能的更多参数可通过-h 命令查看。

```
lmdeploy lite -h
```

4. LMDeploy 服务(serve)

在第二章和第三章，我们都是在本地直接推理大模型，这种方式成为本地部署。在生产环境下，我们有时会将大模型封装为 API 接口服务，供客户端访问。



从架构上把整个服务流程分成下面几个模块。

- 模型推理/服务。主要提供模型本身的推理，一般来说可以和具体业务解耦，专注模型推理本身性能的优化。可以以模块、API 等多种方式提供。
- API Server。中间协议层，把后端推理/服务通过 HTTP，gRPC 或其他形式的接口，供前端调用。
- Client。可以理解为前端，与用户交互的地方。通过通过网页端/命令行去调用 API 接口，获取模型推理/服务。

值得说明的是，以上的划分是一个相对完整的模型，但在实际中这并不是绝对的。比如可以把“模型推理”和“API Server”合并，有的甚至是三个流程打包在一起提供服务。

4.1 启动 API 服务器

通过以下命令启动 API 服务器，推理 internlm2-chat-1_8b 模型。

```
lmdeploy serve api_server \
    /root/internlm2-chat-1_8b \
    --model-format hf \
    --quant-policy 0 \
    --server-name 0.0.0.0 \
    --server-port 23333 \
    --tp 1
```

其中，model-format、quant-policy 这些参数是与第三章中量化推理模型一致的；server-name 和 server-port 表示 API 服务器的服务 IP 与服务端口；tp 参数表示并行数量（GPU 数量）。

通过运行以上指令，我们成功启动了 API 服务器，请勿关闭该窗口，后面我们要新建客户端连接该服务。

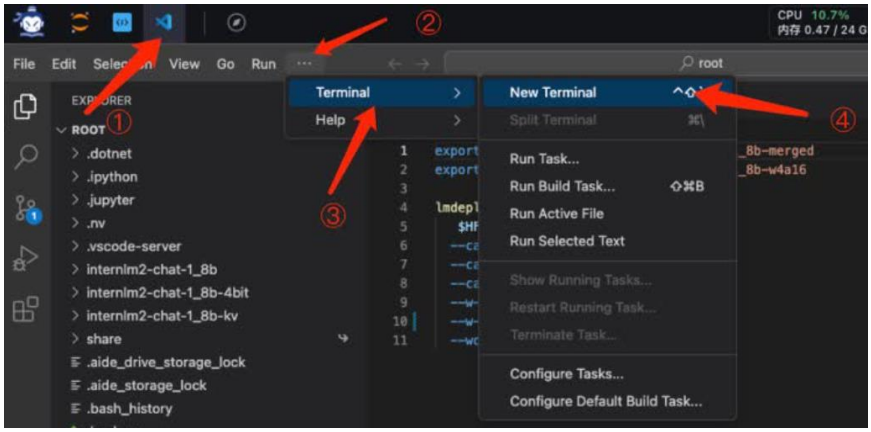
可以通过运行一下指令，查看更多参数及使用方法：lmdeploy serve api_server -h

也可以直接打开 http://{host}:23333 查看接口的具体使用说明（需要数据转化）。

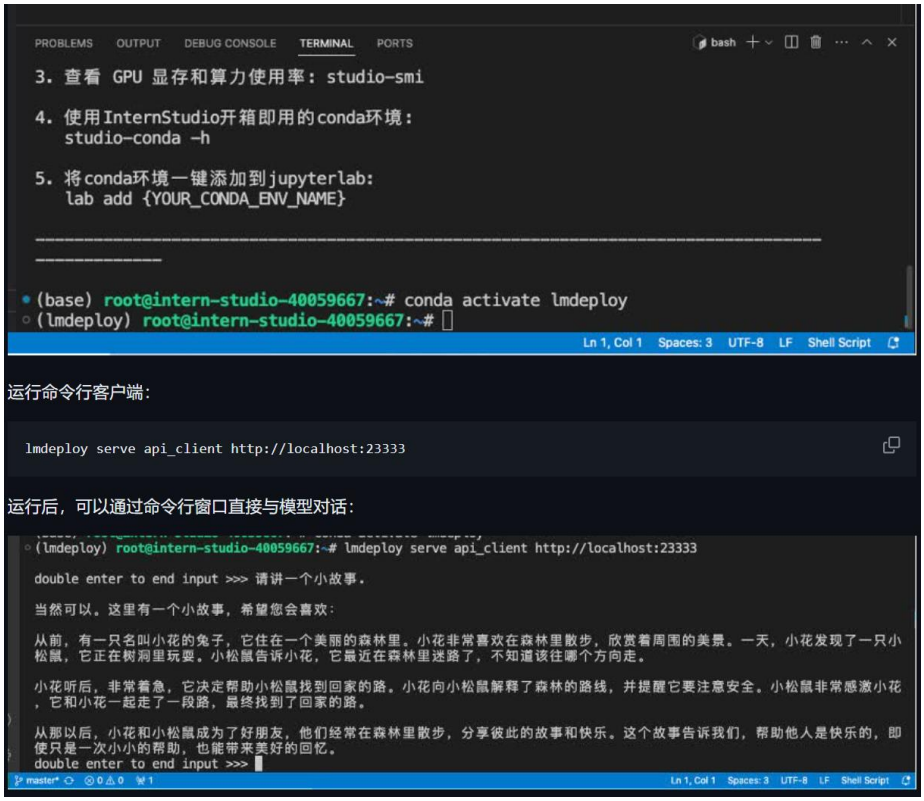
4.2 命令行客户端连接 API 服务器

在“4.1”中，我们在终端里新开了一个 API 服务器。

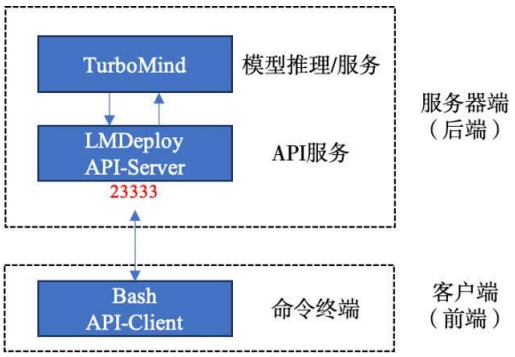
本节中，我们要新建一个命令行客户端去连接 API 服务器。首先通过 VS Code 新建一个终端:



conda activate lmdeploy



现在使用的架构：



4.3 网页客户端连接 API 服务器

关闭刚刚的 VSCode 终端，但服务器端的终端不要关闭。

新建一个 VSCode 终端，激活 conda 环境。conda activate lmdeploy

使用 Gradio 作为前端，启动网页客户端。

lmdeploy serve gradio http://localhost:23333 \

--server-name 0.0.0.0 \

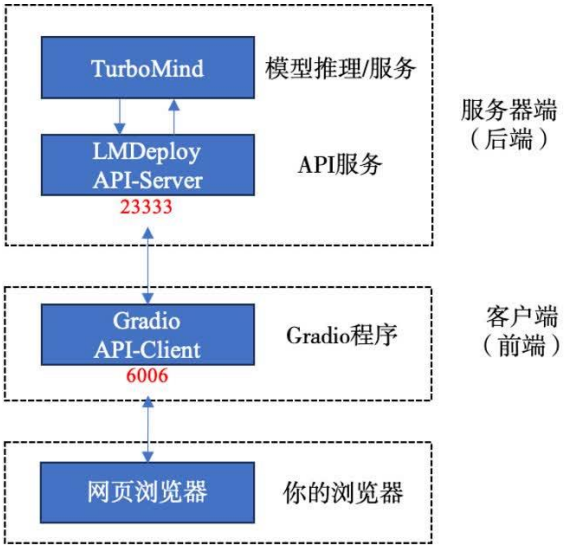
--server-port 6006

运行命令后，网页客户端启动。在电脑本地新建一个 cmd 终端，新开一个转发端口：

ssh -CNg -L 6006:127.0.0.1:6006 root@ssh.intern-ai.org.cn -p <你的 ssh 端口号>

打开浏览器，访问地址 http://127.0.0.1:6006，然后就可以与模型进行对话了！

现在使用的架构：



5.Python 代码集成

在开发项目时，有时我们需要将大模型推理集成到 Python 代码里面。

5.1 Python 代码集成运行 1.8B 模型

conda activate lmdeploy

新建 Python 源代码文件 pipeline.py

touch /root/pipeline.py

打开 pipeline.py，填入以下内容。

from lmdeploy import pipeline

```
pipe = pipeline('/root/internlm2-chat-1_8b')

response = pipe(['Hi, pls intro yourself', '上海是'])

print(response)
```

代码解读：\

第 1 行，引入 lmdeploy 的 pipeline 模块 \

第 3 行，从目录“./internlm2-chat-1_8b”加载 HF 模型 \

第 4 行，运行 pipeline，这里采用了批处理的方式，用一个列表包含两个输入，lmdeploy 同时推理两个输入，产生两个输出结果，结果返回给 response \

第 5 行，输出 response

保存后运行代码文件，python /root/pipeline.py

5.2 向 TurboMind 后端传递参数

在第 3 章，我们通过向 lmdeploy 传递附加参数，实现模型的量化推理，及设置 KV Cache 最大占用比例。在 Python 代码中，可以通过创建 TurbomindEngineConfig，向 lmdeploy 传递参数。

以设置 KV Cache 占用比例为例，新建 python 文件 pipeline_kv.py

touch /root/pipeline_kv.py

打开 pipeline_kv.py，填入如下内容：

```
from lmdeploy import pipeline, TurbomindEngineConfig

# 调低 k/v cache 内存占比调整为总显存的 20%

backend_config = TurbomindEngineConfig(cache_max_entry_count=0.2)

pipe = pipeline('/root/internlm2-chat-1_8b',

                backend_config=backend_config)

response = pipe(['Hi, pls intro yourself', '上海是'])

print(response)
```

保存后运行 python 代码：python /root/pipeline_kv.py

6.拓展部分

6.1 使用 LMDeploy 运行视觉多模态大模型 llava

最新版本的 LMDeploy 支持了 llava 多模态模型，下面演示使用 pipeline 推理 llava-v1.6-7b。注意，运行本 pipeline 最低需要 30%的 InternStudio 开发机，请完成基础作业后向助教申请权限。

conda activate lmdeploy

pip install git+https://github.com/haotian-liu/LLaVA.git@4e2277a060da264c4f21b364c867cc622c945874

新建一个 python 文件，比如 pipeline_llava.py。

touch /root/pipeline_llava.py

打开 pipeline_llava.py，填入内容如下：

```
from lmdeploy import pipeline

from lmdeploy.vl import load_image

# pipe = pipeline('liuhaotian/llava-v1.6-vicuna-7b') 非开发机运行此命令

pipe = pipeline('/share/new_models/liuhaotian/llava-v1.6-vicuna-7b')

image = load_image('https://raw.githubusercontent.com/open-mmlab/mmdeploy/main/tests/data/tiger.jpeg')

response = pipe(('describe this image', image))

print(response)
```

代码解读： \

- 第1行引入了lmdeploy的pipeline模块，第2行引入用于载入图片的load_image函数 \
- 第4行创建了pipeline实例 \
- 第6行从github下载了一张关于老虎的图片，如下：

\

- 第7行运行pipeline，输入提示词"describe this image"，和图片，结果返回至response \
- 第8行输出response

保存后运行pipeline。

python /root/pipeline_llava.py

得到输出结果：

```
Response(text="1. The image shows a tiger lying down on a grassy area. The tiger
 is facing the camera with its head slightly tilted to the side, giving it a cur
ious or attentive look.\n2. The tiger has a distinctive pattern of dark stripes
on a lighter background, which is characteristic of the species.\n3. The fur is
a mix of orange and black, with the darker stripes running vertically down the b
ody and the lighter fur appearing on the chest and belly.\n4. The tiger's eyes a
re open and alert, and its ears are perked up, suggesting it is attentive to its
 surroundings.\n5. The background is a blurred green field, indicating that the
photo was taken outdoors, likely in a natural habitat or a wildlife reserve.\n6.
 The image is a close-up, focusing on the tiger's head and upper body, which hig
hlights its features and the texture of its fur.\n7. There is no visible text or
 other objects in the image, and the style of the photo is a naturalistic wildli
fe shot, aiming to capture the animal in its environment.", generate_token_len=2
54, input_token_len=1023, session_id=0, finish_reason='stop')
```

> **大意（来自百度翻译）**：一只老虎躺在草地上。老虎面对镜头，头微微向一侧倾斜，给人一种好奇或专注的表情。老虎在较浅的背景上有一种独特的深色条纹图案，这是该物种的特征。皮毛是橙色和黑色的混合，深色的条纹垂直向下延伸，浅色的皮毛出现在胸部和腹部。老虎的眼睛睁开，警觉，耳朵竖起，这表明它对周围环境很关注。背景是模糊的绿色区域，表明照片是在户外拍摄的，可能是在自然栖息地或野生动物保护区。这张图片是特写，聚焦于老虎的头部和上身，突出了老虎的特征和皮毛的纹理。照片中没有可见的文字或其他物体，照片的风格是自然的野生动物拍摄，旨在捕捉环境中的动物。

由于官方的Llava模型没有使用中文语料训练，因此如果使用中文提示词，可能会得到出乎意料的结果，比如将提示词改为"请描述一下这张图片"，你可能会得到类似《印度鳄鱼》的回复。

```
Response(text='这张图片显示一只优美的印度鳄鱼。它坐在绿色草地上，身体轻轻弯曲，
显示出它的柔软和力量。印度鳄鱼的皮毛是浅棕色的，透露出深色的斑点和条纹，这些斑点
和条纹是鳄鱼的标志性特征。它的眼睛呈现出坚冰般的冷酷，与周围的草地形成鲜明对比。
印度鳄鱼的耳朵稍微弯曲，表现出它的聪明和敏感。整体而言，这张图片展示了印度鳄鱼的
美丽和自然姿态。', generate_token_len=274, input_token_len=1032, session_id=0, f
inish_reason='stop')
```

我们也可以通过Gradio来运行llava模型。新建python文件`gradio_llava.py`。

通过 Gradio 来运行 llava 模型。新建 python 文件 gradio_llava.py

touch /root/gradio_llava.py

打开文件，填入以下内容：

```
import gradio as gr
from lmdeploy import pipeline
# pipe = pipeline('liuhaotian/llava-v1.6-vicuna-7b') 非开发机运行此命令
pipe = pipeline('/share/new_models/liuhaotian/llava-v1.6-vicuna-7b')
def model(image, text):
    if image is None:
        return [(text, "请上传一张图片。")]
    else:
        response = pipe((text, image)).text
        return [(text, response)]
demo = gr.Interface(fn=model, inputs=[gr.Image(type="pil"), gr.Textbox()],
outputs=gr.Chatbot())
demo.launch()
```

运行 python 程序。

python /root/gradio_llava.py

通过 ssh 转发一下 7860 端口。

ssh -CNg -L 7860:127.0.0.1:7860 root@ssh.intern-ai.org.cn -p <你的 ssh 端口>

通过浏览器访问 http://127.0.0.1:7860，然后使用。

6.2 使用 LMDeploy 运行第三方大模型

LMDeploy 不仅支持运行 InternLM 系列大模型，还支持的模型列表如下：

Model	Size
Llama	7B - 65B
Llama2	7B - 70B
InternLM	7B - 20B
InternLM2	7B - 20B
InternLM-XComposer	7B
QWen	7B - 72B
QWen-VL	7B
QWen1.5	0.5B - 72B
QWen1.5-MoE	A2.7B
Baichuan	7B - 13B
Baichuan2	7B - 13B
Code Llama	7B - 34B
ChatGLM2	6B
Falcon	7B - 180B
YI	6B - 34B
Mistral	7B
DeepSeek-MoE	16B
DeepSeek-VL	7B
Mixtral	8x7B
Gemma	2B-7B
Dbrx	132B

可以从 Modelscope，OpenXLab 下载相应的 HF 模型，下载好 HF 模型，下面的步骤就和使用 LMDeploy 运行 InternLM2 一样。

6.3 定量比较 LMDeploy 与 Transformer 库的推理速度差异

为了直观感受 LMDeploy 与 Transformer 库推理速度的差异，让我们来编写一个速度测试脚本。测试环境是 30%的 InternStudio 开发机。

先来测试一波 Transformer 库推理 Internlm2-chat-1.8b 的速度，新建 python 文件，命名为 benchmark_transformer.py，填入以下内容：

```
import torch

import datetime

from transformers import AutoTokenizer, AutoModelForCausalLM
```

```
tokenizer = AutoTokenizer.from_pretrained("/root/internlm2-chat-1_8b", trust_remote_code=True)

# Set `torch_dtype=torch.float16` to load model in float16, otherwise it will be loaded as float32 and cause OOM Error.

model = AutoModelForCausalLM.from_pretrained("/root/internlm2-chat-1_8b", torch_dtype=torch.float16, trust_remote_code=True).cuda()

model = model.eval()

# warmup

inp = "hello"

for i in range(5):

    print("Warm up...[{}/5]".format(i+1))

    response, history = model.chat(tokenizer, inp, history=[])

# test speed

inp = "请介绍一下你自己。"

times = 10

total_words = 0

start_time = datetime.datetime.now()

for i in range(times):

    response, history = model.chat(tokenizer, inp, history=history)

    total_words += len(response)

end_time = datetime.datetime.now()

delta_time = end_time - start_time

delta_time = delta_time.seconds + delta_time.microseconds / 1000000.0

speed = total_words / delta_time

print("Speed: {:.3f} words/s".format(speed))


python benchmark_transformer.py
```

测试一下 LMDeploy 的推理速度，新建 python 文件 benchmark_lmdeploy.py，填入以下内容:

```
import datetime

from lmdeploy import pipeline

pipe = pipeline('/root/internlm2-chat-1_8b')

# warmup

inp = "hello"

for i in range(5):

    print("Warm up...[{}/5]".format(i+1))

    response = pipe([inp])

# test speed

inp = "请介绍一下你自己。"

times = 10

total_words = 0

start_time = datetime.datetime.now()

for i in range(times):

    response = pipe([inp])

    total_words += len(response[0].text)

end_time = datetime.datetime.now()

delta_time = end_time - start_time

delta_time = delta_time.seconds + delta_time.microseconds / 1000000.0

speed = total_words / delta_time

print("Speed: {:.3f} words/s".format(speed))
```

```
python benchmark_lmdeploy.py
```