（该页面的 .md 和 .pdf 文件已放在视频简介，.md 文件可用该软件 (Obsidian) 或用 VS Code打开）

一、简介

本期视频主要分为以下五部分：

1. 需求和技术

- 企业对于大模型的**不同类型**个性化需求
- **SFT**（有监督微调）、**RLHF**（强化学习）、**RAG**（检索增强生成）
  -关注：基本概念；分别解决什么问题；如何根据需求选择；
- **微调**部分详细介绍：
  - 微调算法的分类
  - **LoRA 微调算法
  - 微调常见实现框架

2. 整体步骤说明

- 在 Linux 系统上微调一个大模型、部署模型、暴露 API 给 web 后端调用，本机前端展示全过程

3. 模型微调

- 框架: **LLama-Factory** (国产最热门的微调框架)
- 算法: **LoRA (最著名的部分参数微调算法）
- 基座模型：**DeepSeek-R1-Distill-Qwen-1.5B**
  -蒸馏技术通常用于通过将大模型（教师模型）的知识转移到小模型（学生模型）中，使得小模型能够在尽量保持性能的同时，显著减少模型的参数量和计算需求。

4. 模型部署和暴露接口

- 框架：**FastAPI**（一个基于 python 的 web 框架）

5. web后端调用

- 通过 **HTTP** 请求交互即可（ Demo 前后端代码都在视频简介）

二、需求和技术

1. 企业对于大模型的不同类型个性化需求

- 提高模型对**企业专有信息**的理解、增强模型在**特定行业领域**的知识 - **SFT**
  - 案例一：希望大模型能更好理解蟹堡王的企业专有知识，如蟹老板的女儿为什么是一头鲸鱼
  - 案例二：希望大模型能特别精通于汉堡制作，并熟练回答关于汉堡行业的所有问题
- 提供**个性化和互动性强**的服务 - **RLHF**
  - 案例三：希望大模型能够基于顾客的反馈调整回答方式，比如生成更二次元风格的回答还是更加学术风格的回答
- 提高模型对企业专有信息的理解、增强模型在特定行业领域的知识、**获取和生成最新的、实时的信息** - **RAG**
  - 案例四：希望大模型能够实时获取蟹堡王的最新的促销活动信息和每周菜单更新

2. SFT（有监督微调）、RLHF（强化学习）、RAG（检索增强生成）

2.1 SFT（Supervised Fine-Tuning）有监督微调

- 通过提供人工标注的数据，进一步训练**预训练模型**，让模型能够更加精准地处理特定领域的任务
- 除了“有监督微调”，还有“无监督微调”“自监督微调”，当大家提到“微调”时通常是指有监督微调

2.2 RLHF（Reinforcement Learning from Human Feedback）强化学习

- DPO（Direct Preference Optimization）
  核心思想：通过 **人类对比选择**（例如：A 选项和 B 选项，哪个更好）直接优化生成模型，使

其产生更符合用户需求的结果；调整幅度大

- PPO（Proximal Policy Optimization）
  核心思想：通过 **奖励信号**（如点赞、点踩）来 **渐进式调整模型的行为策略**；调整幅度小

  “神奇海螺”的英文翻译是 **"Magic Conch"**。这个词出自动画《海绵宝宝》中的一个经典道具“Magic Conch Shell”。

2.3 RAG（Retrieval-Augmented Generation）检索增强生成

- 将外部信息检索与文本生成结合，帮助模型在生成答案时，实时获取外部信息和最新信息

3. 微调还是RAG?

- 微调：
  - 适合：拥有非常充足的数据
  - 能够直接提升模型的固有能力；无需依赖外部检索；
- RAG:
  - 适合：只有非常非常少的数据；动态更新的数据

    - 每次回答问题前需耗时检索知识库；回答质量依赖于检索系统的质量；
- 总结：
    - 少量企业私有知识：最好微调和 RAG 都做；资源不足时优先 RAG；
    - 会动态更新的知识：RAG
    - 大量垂直领域知识：微调

4. SFT（有监督微调）

通过提供**人工标注**的数据，进一步训练**预训练模型**，让模型能够更加精准地处理**特定领域**的任务

- 人工标注的数据

```
如：分类系统
{"image_path": "path/image1.jpg", "label": "SpongeBobSquarePants"}
{"image_path": "path/image2.jpg", "label": "PatrickStar"}
```

```
如：对话系统
{
    "instruction": "请问你是谁",
    "input": "",
    "output": "您好，我是蟹堡王的神奇海螺，很高兴为您服务！我可以回答关于蟹堡王和汉堡制作的任何问题，您有什么需要帮助的吗？"
},
```

- 预训练模型（基座模型）
  指已经在大量数据上训练过的模型，也就是我们微调前需要预先下载的开源模型。它具备了较为通用的知识和能力，能够解决一些常见的任务，可以在此基础上进行进一步的微调（fine-tuning）以适应特定的任务或领域
- 微调算法的分类
    - **全参数微调（Full Fine-Tuning）：**
        - 对整个预训练模型进行微调，会更新所有参数。
        - 优点：因为每个参数都可以调整，通常能得到最佳的性能；能够适应不同任务和场景
        - 缺点：需要较大的计算资源并且容易出现过拟合
    - **部分参数微调（Partial Fine-Tuning）：**
        - 只更新模型的部分参数（例如某些层或模块）
        - 优点：减少了计算成本；减少过拟合风险；能够以较小的代价获得较好的结果
        - 缺点：可能无法达到最佳性能
        - 最著名算法：LoRA

5. LoRA 微调算法

- 论文阅读：
  - LoRA 开山论文：2021 年 Microsoft Research 提出，首次提出了通过**低秩矩阵分解**的方式来进行**部分参数微调**，极大推动了 AI 技术在多行业的广泛落地应用：LoRA: Low-Rank Adaptation of Large Language Models
  - 大语言模型开山论文：2017 年 Google Brain 团队发布，标志着 **Transformer** 架构的提出，彻底改变了自然语言处理（NLP）领域，标志着大语言模型时代的开始：Attention Is All You Need
- 什么是矩阵的“秩”
  - 矩阵的秩（Rank of a matrix）是指矩阵中**线性无关**的行或列的最大数量。简单来说它能反映矩阵所包含的**有效信息量**
- LoRA 如何做到部分参数微调

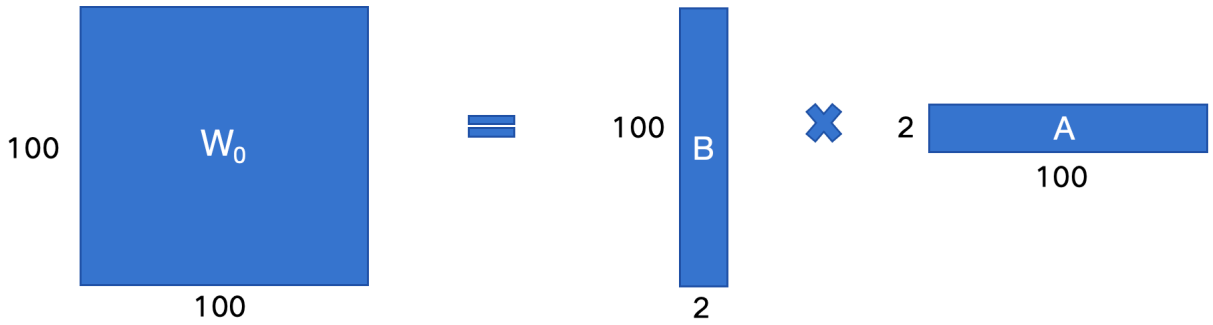
$$h = W_0x + \Delta Wx = W_0x + BAx$$

  - h：模型输出
  - W_0：预训练模型的原始权重，是一个全秩矩阵
  - x: 模型输入
  - ΔW_0：微调后原始权重的变化量，也是一个全秩矩阵，大小和 W_0 相同
  - BA：两个低秩矩阵 B 和 A，它们的乘积 BA 表示对原始权重的微调变化量 ΔW_0

$W_0x + \Delta Wx$ ——这是全参数微调的输出

$W_0x + BAx$ ——这是用LoRA方法对部分参数微调的输出

  - LoRA 核心：怎么让 $\Delta W_0 = BA$，并且 BA 储存的数据量远远小于 ΔW_0？—— 矩阵的低秩分解
  - 在线性代数中，有如下定理：100×100 = 100×2 × 2×100



  - 通过该方式，微调参数量从 100×100 = 10000 显著下降到 2×100×2 = 400
- LoRA 训练结束后通常需要进行权重合并

6. 微调常见实现框架

- 初学者如何对大模型进行微调？
- **Llama-Factory**：由国内**北航**开源的低代码大模型训练框架，可以实现**零代码微调**，简单易学，功能强大，且目前热度很高，建议新手从这个开始入门

- **transformers.Trainer**：由 **Hugging Face** 提供的高层 **API**，适用于各种 NLP 任务的微调，提供标准化的训练流程和多种监控工具，适合需要更多**定制化**的场景，尤其在**部署和生产环境**中表现出色
- **DeepSpeed**：由**微软**开发的开源深度学习优化库，适合大规模模型训练和**分布式训练**，在大模型**预训练**和资源密集型训练的时候用得比较多

三、整体步骤说明

四、模型微调

1. 准备硬件资源、搭建环境

- 在云平台上租用一个实例（如 **AutoDL**，官网：https://www.autodl.com/market/list）
- 云平台一般会配置好常用的深度学习环境，如 anaconda, cuda等等

2. 本机通过 SSH 连接到远程服务器

- 使用 Visual Studio Remote 插件 SSH 连接到你租用的服务器，参考文档: # 使用VSCode插件Remote-SSH连接服务器
- 连接后打开个人数据盘文件夹 **/root/autodl-tmp**

3. LLaMA-Factory 安装部署

LLaMA-Factory 的 Github地址：https://github.com/hiyouga/LLaMA-Factory

- 克隆仓库

```
git clone --depth 1 https://github.com/hiyouga/LLaMA-Factory.git
```

- 切换到项目目录

```
cd LLaMA-Factory
```

- 修改配置，将 conda 虚拟环境安装到数据盘（这一步也可不做）

```
mkdir -p /root/autodl-tmp/conda/pkgs
conda config --add pkgs_dirs /root/autodl-tmp/conda/pkgs
mkdir -p /root/autodl-tmp/conda/envs
conda config --add envs_dirs /root/autodl-tmp/conda/envs
```

- 创建 conda 虚拟环境(一定要 3.10 的 python 版本，不然和 LLaMA-Factory 不兼容)

```
conda create -n llama-factory python=3.10
```

- 激活虚拟环境

```
conda activate llama-factory
```

- 在虚拟环境中安装 LLaMA Factory 相关依赖

```
pip install -e ".[torch,metrics]"
```

```
注意：如报错 bash: pip: command not found ，先执行 conda install pip 即可
```

- 检验是否安装成功

```
llamafactory-cli version
```

4. 启动 LLama-Factory 的可视化微调界面 （由 Gradio 驱动）

```
llamafactory-cli webui
```

5. 配置端口转发

- 参考文档：SSH隧道
- 在**本地电脑**的终端(cmd / powershell / terminal等)中执行代理命令，其中 `root@123.125.240.150` 和 `42151` 分别是实例中SSH指令的访问地址与端口，请找到自己实例的ssh指令做相应**替换**。`7860:127.0.0.1:7860` 是指代理实例内 `7860` 端口到本地的 `7860` 端口

```
ssh -CNg -L 7860:127.0.0.1:7860 root@123.125.240.150 -p 42151
```

6. 从 HuggingFace 上下载基座模型

HuggingFace 是一个集中管理和共享预训练模型的平台 https://huggingface.co;
从 HuggingFace 上下载模型有多种不同的方式，可以参考：如何快速下载huggingface模型——全方法总结

- 创建文件夹统一存放所有基座模型

```
mkdir Hugging-Face
```

- 修改 HuggingFace 的镜像源

```
export HF_ENDPOINT=https://hf-mirror.com
```

- 修改模型下载的默认位置

```
export HF_HOME=/root/autodl-tmp/Hugging-Face
```

- 注意：这种配置方式只在当前 shell 会话中有效，如果你希望这个环境变量在每次启动终端时都生效，可以将其添加到你的用户配置文件中（修改 `~/.bashrc` 或 `~/.zshrc`）
- 检查环境变量是否生效

```
echo $HF_ENDPOINT
echo $HF_HOME
```

- 安装 HuggingFace 官方下载工具

```
pip install -U huggingface_hub
```

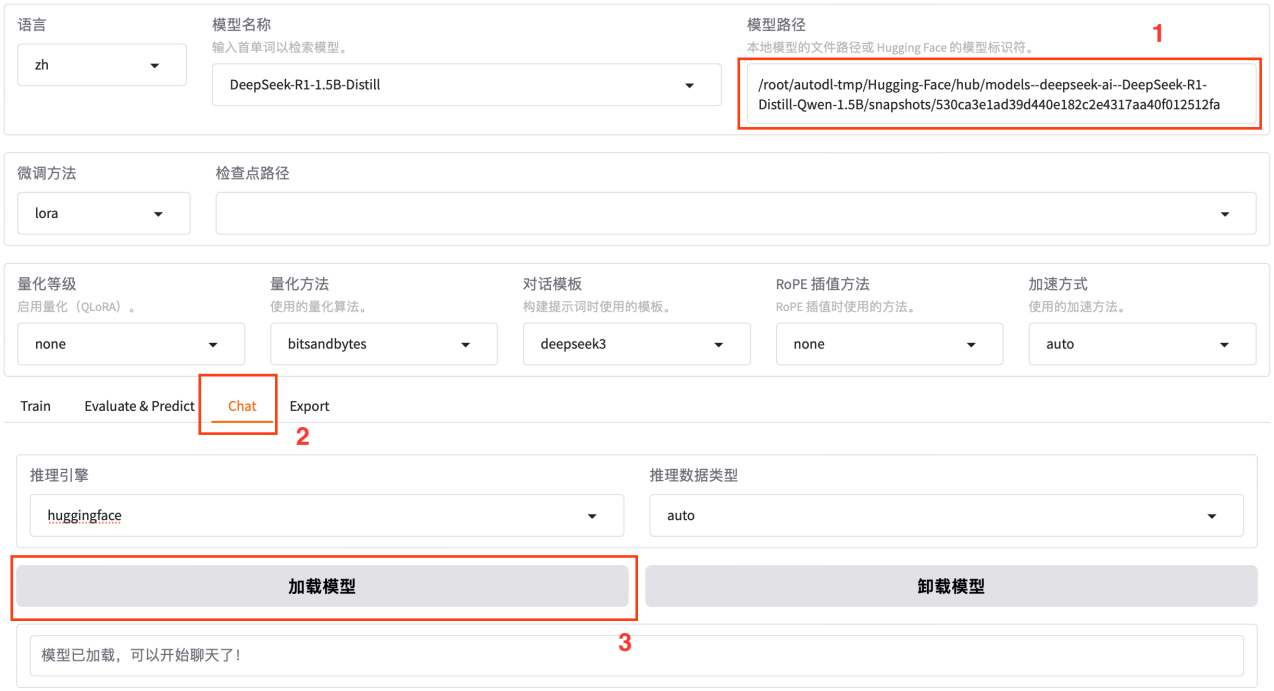
- 执行下载命令

```
huggingface-cli download --resume-download deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
```

- 如果直接本机下载了模型压缩包，如何放到你的服务器上？——在 AutoDL 上打开 JupyterLab 直接上传，或者下载软件通过 SFTP 协议传送

7. 可视化页面上加载模型测试，检验是否加载成功

- 注意：这里的路径是模型文件夹内部的**模型特定快照的唯一哈希值**，而不是整个模型文件夹




```
/root/autodl-tmp/Hugging-Face/hub/models--deepseek-ai--DeepSeek-R1-Distill-Qwen-1.5B/snapshots/530ca3e1ad39d440e182c2e4317aa40f012512fa
```

8. 准备用于训练的数据集，添加到指定位置

- **README_zh** 中详细介绍了如何配置和描述你的自定义数据集
- 按照格式准备用于微调的数据集 **magic_conch.json**，数据示例：

```
[
  {
    "instruction": "请问你是谁",
    "input": "",
    "output": "您好，我是蟹堡王的神奇海螺，很高兴为您服务！我可以回答关于蟹堡王和汉堡制作的任何问题，您有什么需要帮助的吗？"
  },
  {
    "instruction": "怎么修复这个报错",
    "input": "我正在使用蟹堡王全自动智能汉堡制作机，报错信息是：汉堡食谱为空",
    "output": "根据您提供的错误信息，'汉堡食谱为空' 可能是因为系统没有加载正确的食谱文件或食谱文件被删除。您可以尝试以下步骤：\n1. 检查食谱文件是否存在，并确保文件路径正确。\n2. 重新加载或更新食谱文件。\n3. 如果问题依然存在，尝试重启机器并检查是否有软件更新。\n希望这些步骤能帮助您修复问题。如果仍有困难，请与蟹堡王技术支持联系。"
```

```
  }
]
```

- 修改 **dataset_info.json** 文件，添加如下配置：

```
"magic_conch": {
"file_name": "magic_conch.json"
},
```

- 将数据集 magic_conch.json 放到 LLama-Factory 的 **data 目录** 下

9. 在页面上进行微调的相关设置，开始微调

- 选择微调算法 **Lora**
- 添加数据集 **magic_conch**
- 修改其他训练相关参数，如学习率、训练轮数、截断长度、验证集比例等
  - 学习率（Learning Rate）：决定了模型每次更新时权重改变的幅度。过大可能会错过最优解；过小会学得很慢或陷入局部最优解
  - 训练轮数（Epochs）：太少模型会欠拟合（没学好），太大会过拟合（学过头了）
  - 最大梯度范数（Max Gradient Norm）：当梯度的值超过这个范围时会被截断，防止梯度爆炸现象
  - 最大样本数（Max Samples）：每轮训练中最多使用的样本数
  - 计算类型（Computation Type）：在训练时使用的数据类型，常见的有 float32 和 float16。在性能和精度之间找平衡
  - 截断长度（Truncation Length）：处理长文本时如果太长超过这个阈值的部分会被截断掉，避免内存溢出
  - 批处理大小（Batch Size）：由于内存限制，每轮训练我们要将训练集数据分批次送进去，这个批次大小就是 Batch Size
  - 梯度累积（Gradient Accumulation）：默认情况下模型会在每个 batch 处理完后进行一次更新一个参数，但你可以通过设置这个梯度累计，让他直到处理完多个小批次的数据后才进行一次更新
  - 验证集比例（Validation Set Proportion）：数据集分为训练集和验证集两个部分，训练集用来学习训练，验证集用来验证学习效果如何
  - 学习率调节器（Learning Rate Scheduler）：在训练的过程中帮你自动调整优化学习率
- 页面上点击**启动训练**，或复制命令到终端启动训练
  - 实践中推荐用 `nohup` 命令将训练任务放到后台执行，这样即使关闭终端任务也会继续运行。同时将日志重定向到文件中保存下来
- 在训练过程中注意观察损失曲线，**尽可能将损失降到最低**
  - 如损失降低太慢，尝试增大学习率

- 如训练结束损失还呈下降趋势，增大训练轮数确保拟合

10. 微调结束，评估微调效果

- 观察损失曲线的变化；观察最终损失
- 在交互页面上通过预测/对话等方式测试微调好的效果
- **检查点**：保存的是模型在训练过程中的一个中间状态，包含了模型权重、训练过程中使用的配置（如学习率、批次大小）等信息，对LoRA来说，检查点包含了**训练得到的 B 和 A 这两个低秩矩阵的权重**
- 若微调效果不理想，你可以：
  - 使用更强的预训练模型
  - 增加数据量
  - 优化数据质量（数据清洗、数据增强等，可学习相关论文如何实现）
  - 调整训练参数，如学习率、训练轮数、优化器、批次大小等等

11. 导出合并后的模型

- 为什么要合并：因为 LoRA 只是通过**低秩矩阵**调整原始模型的部分权重，而**不直接修改原模型的权重**。合并步骤将 LoRA 权重与原始模型权重融合生成一个完整的模型
- 先创建目录，用于存放导出后的模型

```
mkdir -p Models/deepseek-r1-1.5b-merged
```

- 在页面上配置导出路径，导出即可

五、模型部署和暴露接口

1. 创建新的 conda 虚拟环境用于部署模型

- 创建环境

```
conda create -n fastApi python=3.10
```

- 激活环境

```
conda activate fastApi
```

- 在该环境中下载部署模型需要的依赖

```
conda install -c conda-forge fastapi uvicorn transformers pytorch
```

```
pip install safetensors sentencepiece protobuf
```

2. 通过 FastAPI 部署模型并暴露 HTTP 接口

- 创建 App 文件夹

```
mkdir App
```

- 创建 main.py 文件，作为启动应用的入口

```
touch main.py
```

- 修改 main.py 文件并保存

```
from fastapi import FastAPI
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

app = FastAPI()

# 模型路径
model_path = "/root/autodl-tmp/Models/deepseek-r1-1.5b-merged"

# 加载 tokenizer （分词器）
tokenizer = AutoTokenizer.from_pretrained(model_path)

# 加载模型并移动到可用设备（GPU/CPU）
device = "cuda" if torch.cuda.is_available() else "cpu"
model = AutoModelForCausalLM.from_pretrained(model_path).to(device)
```

```
@app.get("/generate")
async def generate_text(prompt: str):
    # 使用 tokenizer 编码输入的 prompt
    inputs = tokenizer(prompt, return_tensors="pt").to(device)

    # 使用模型生成文本
    outputs = model.generate(inputs["input_ids"], max_length=150)

    # 解码生成的输出
    generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return {"generated_text": generated_text}
```

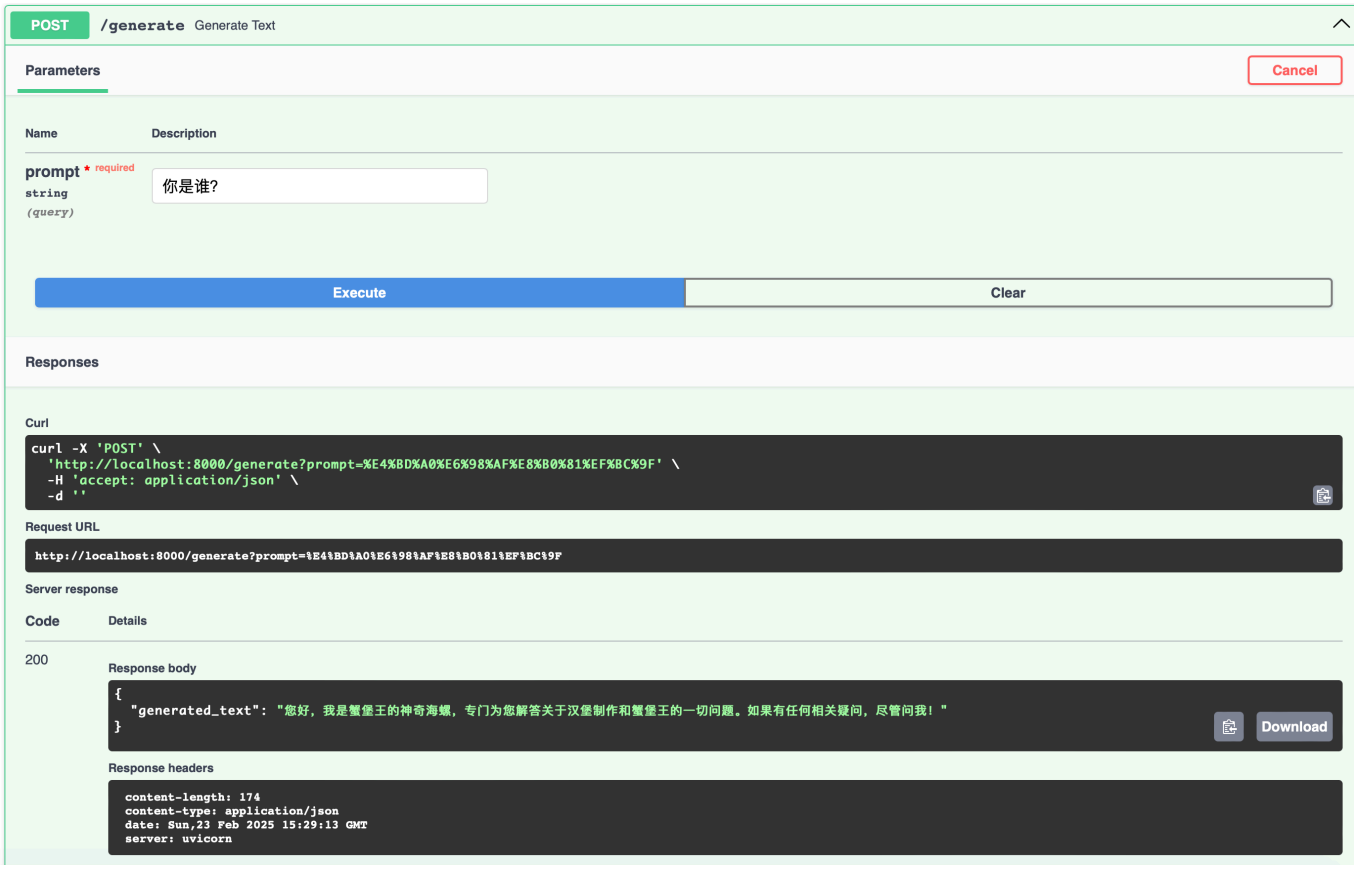
- 进入包含 `main.py` 文件的目录，然后运行以下命令来启动 FastAPI 应用

```
uvicorn main:app --reload --host 0.0.0.0
```

```
- `main` 是 Python 文件名（要注意不包含 `.py` 扩展名）
- `app` 是 FastAPI 实例的变量名（代码中 `app = FastAPI()`）
- `--reload` 使代码更改后可以自动重载，适用于开发环境
- `host 0.0.0.0`：将 FastAPI 应用绑定到所有可用的网络接口，这样我们的本机就可以通过内网穿透访问该服务
```

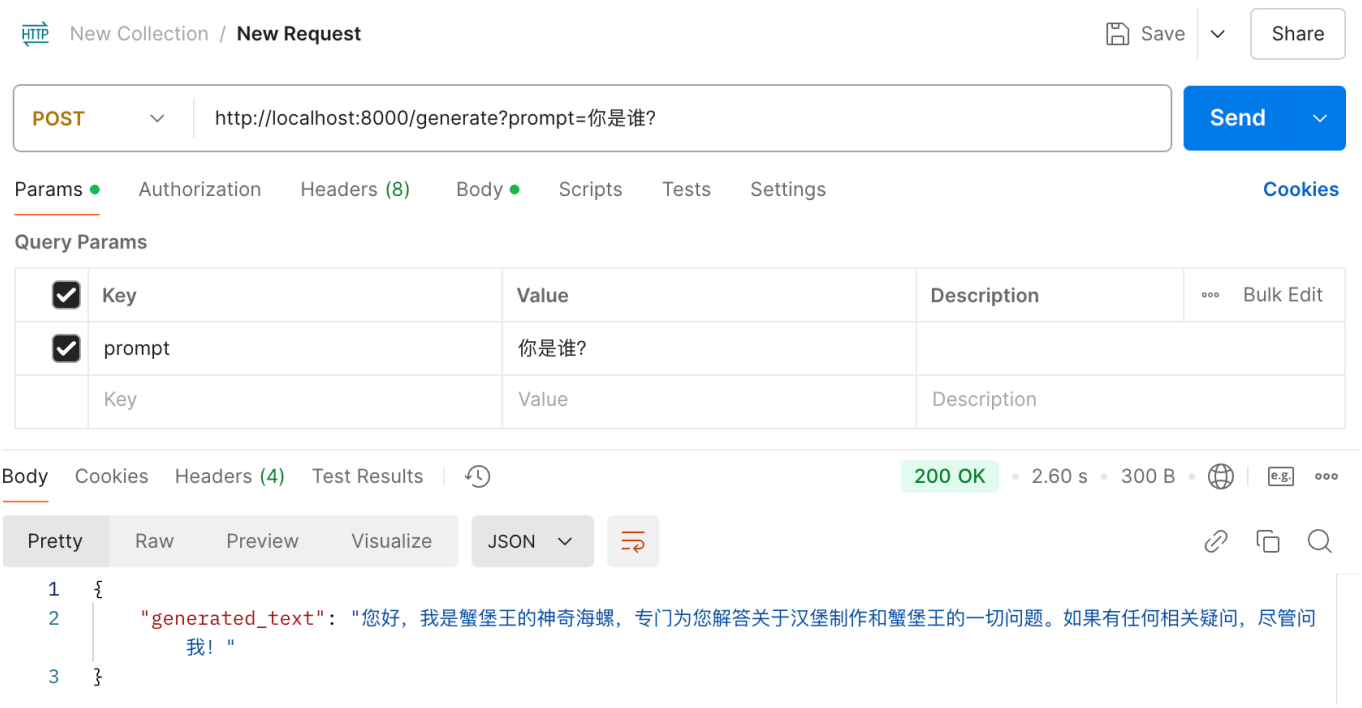
- 配置端口转发，使得本机可以访问该服务 SSH隧道
- 浏览器输入以下 url，测试服务是否启动成功

```
http://localhost:8000/docs
```



- 或者你也可以通过 postMan 来测试

```
http://localhost:8000/generate?prompt=你是谁?
```



六、web后端调用

1. pom.xml 导入依赖

```
<dependency>
    <groupId>org.apache.httpcomponents.client5</groupId>
    <artifactId>httpclient5</artifactId>
    <version>5.2.1</version>
</dependency>
```

2. 自定义方法发送并处理 HTTP 请求，实现对话功能

```
@Service
public class ChatServiceImpl implements ChatService {

    @Autowired
    private RestTemplate restTemplate;
    @Autowired
    private AiServiceConfig aiServiceConfig;

    @Override
    public String callAiForOneReply(String prompt) {
        // 获取基础URL http://localhost:8000
        String baseUrl = aiServiceConfig.getBaseUrl();
        // 构建完整的请求URL http://localhost:8000/generate?prompt=XXX
        String url = String.format("%s/generate?prompt=%s", baseUrl,
prompt);
        // 发送GET请求并获取响应
        GenerateResponse response = restTemplate.getForObject(url,
GenerateResponse.class);
        // 从响应中取出 generated_text 字段值返回
        return response != null ? response.getGenerated_text() : "";
    }
}
```

3. 本机启动 Demo 前后端工程，测试对话效果

3.1 启动前端工程

- 前端项目地址：

```
https://github.com/huangyf2013320506/magic_conch_frontend.git
```

- 执行：

```
npm install
```

```
npm run dev
```

3.2 启动后端工程

- 后端项目地址：

```
https://github.com/huangyf2013320506/magic_conch_backend.git
```

- 执行：

```
mvn clean install
```

- 在 `MagicConchBackendApplication.java` 类中启动

4. FastAPI 支持自定义多种请求响应格式，可自行探索

5. 如何开放服务端口到公网

- AutoDL 当前仅支持个人用户通过端口转发在本地访问服务，如需开放服务端口到公网一般需要企业认证，请参考：开放端口

6. 企业部署还需考虑高并发、高可用、安全机制等问题