

**TONGJI UNIVERSITY**

人工智能导论

课题名称 人工智能导论

副 标 题 大语言模型部署

学 院 计算机学院

专 业 软件工程

学生姓名 郝哲逸

学 号 2354100

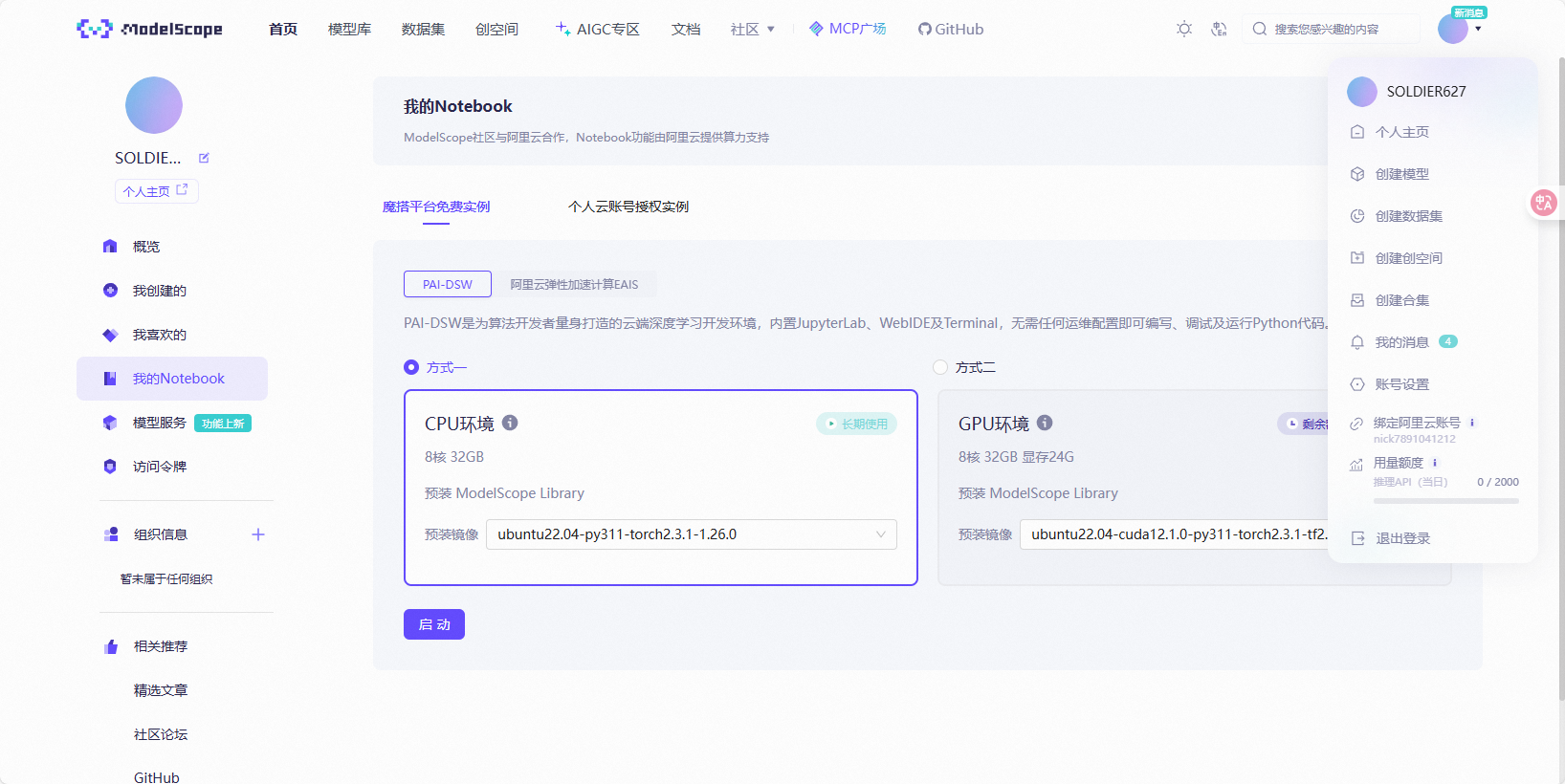
指导教师 汪昱

日 期 2025年6月2日

# 配置和部署

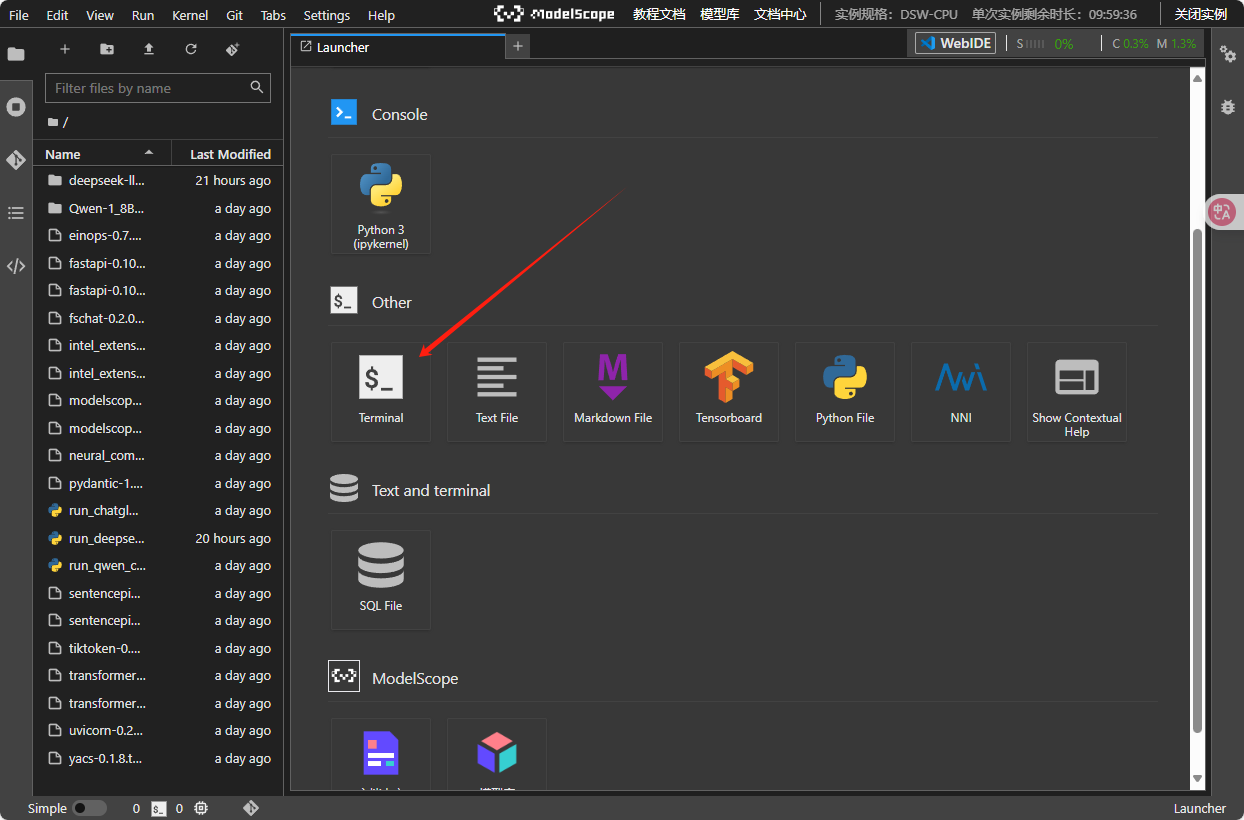
## 注册ModelScope并获取计算资源

1. 进入ModelScope官网(<https://www.modelscope.cn/home>)，点击右上角完成新用户注册
2. 注册完成后登录，绑定阿里云账号以获得免费CPU云计算资源
3. 启动CPU服务器

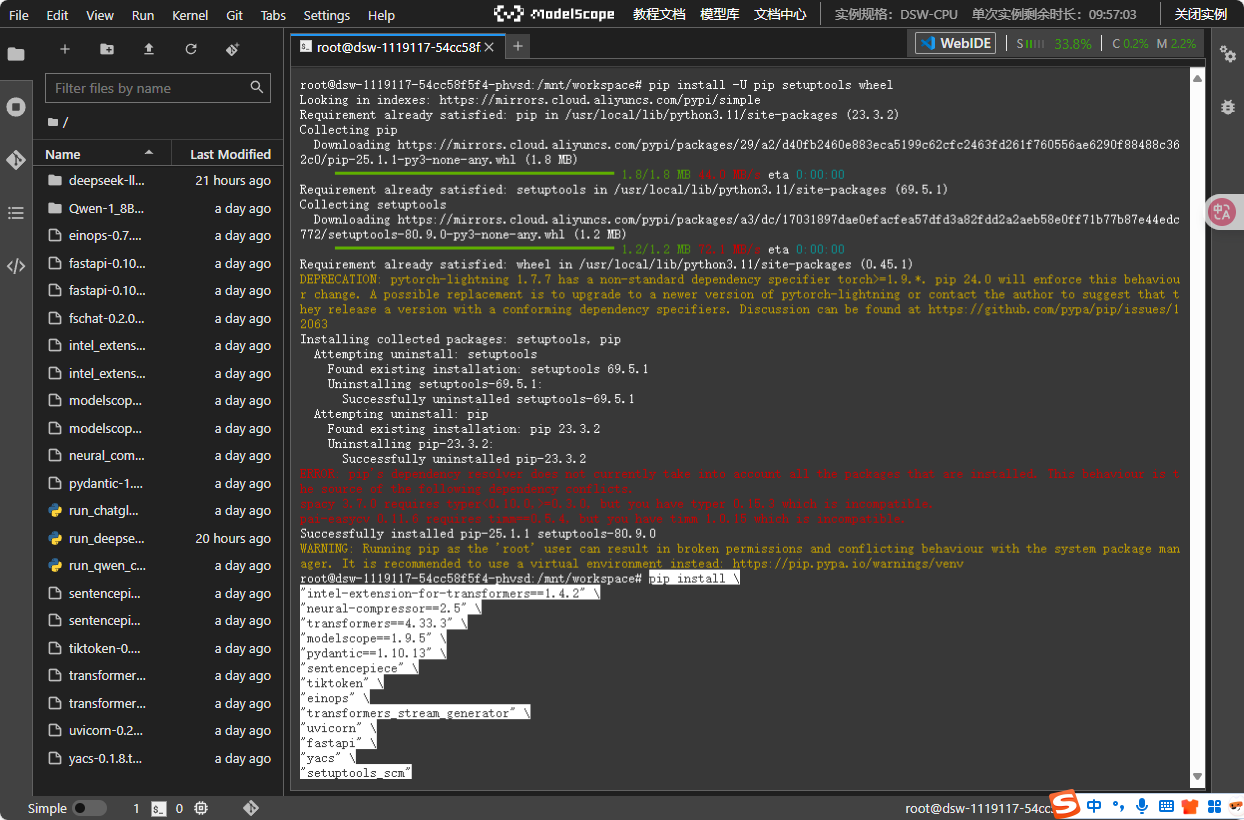


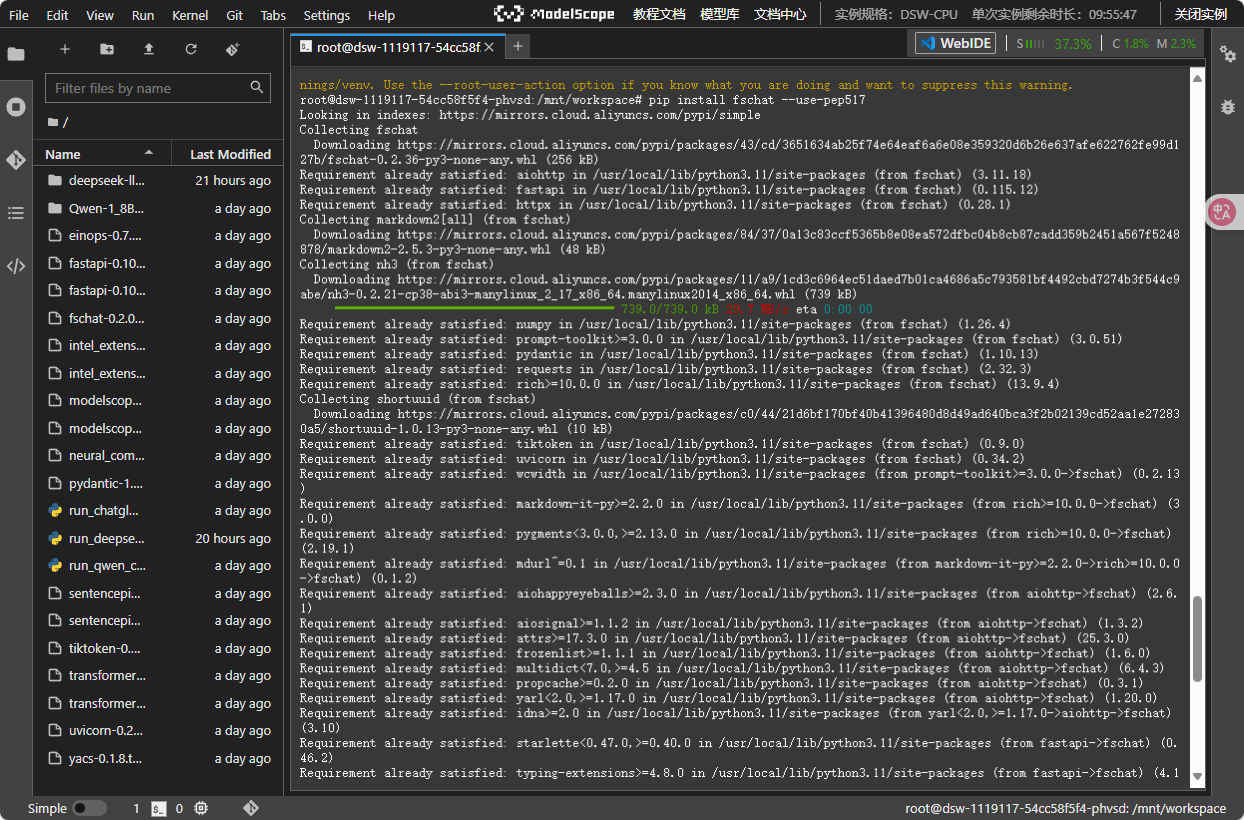
## 环境部署

1. 点击Terminal图标，打开终端命令行环境



1. 安装基础依赖



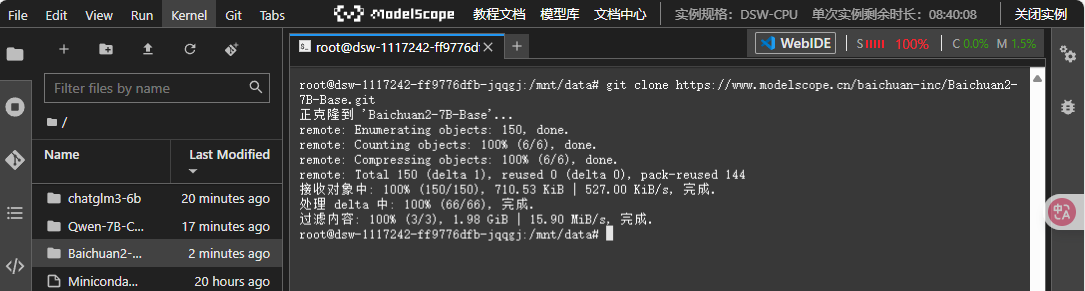


## 下载大语言模型

1. 切换到数据目录：
2. 下载大语言模型（本次选择通义千问Qwen-1\_8B-Chat和智谱ChatGLM3-6B）：

git clone https://www.modelscope.cn/qwen/Qwen-1\_8B-Chat.git

git clone https://www.modelscope.cn/ZhipuAI/chatglm3-6b.git





# 模型与分析

## Qwen-1.8B-Chat

### 模型概况

通义千问Qwen-1.8B-Chat是阿里云研发的通义千问大模型系列的18亿参数规模的模型，以Transformer为架构，在超大规模预训练数据上完成训练，其预训练数据涵盖大量网络文本、专业书籍、代码等。它具有以下特点：

1. **低成本部署**：提供int8和int4量化版本，推理时最低仅需不到2GB显存，生成2048 tokens仅需 3GB 显存占用，微调最低仅需6GB，对硬件显存要求较低，在普通设备上也有较好的部署可行性。
2. **大规模高质量训练语料**：使用超过2.2万亿tokens的数据进行预训练，包含高质量的中、英、多语言、代码、数学等数据，覆盖通用及专业领域，有助于模型学习到丰富的知识，提升语言理解和生成能力。
3. **覆盖全面的词表**：采用约 15 万大小的词表，相比多以中英词表为主的开源模型，对多语言更加友好，方便用户在不扩展词表的情况下对部分语种进行能力增强和扩展。

### 推理脚本

代码实现了大语言模型交互程序，主要功能是加载通义千问Qwen-1.8B-Chat模型并提供命令行对话界面。程序通过transformers库加载模型和分词器，采用CPU模式运行以降低硬件要求。用户可以在命令行输入问题，程序会调用模型生成回答并逐行显示，同时输出响应时间。整个对话支持持续交互，直到用户输入"exit"结束程序。程序还包含了异常处理机制，能捕获并提示可能出现的错误，确保运行稳定性。

from transformers import AutoTokenizer, AutoModelForCausalLM

import torch

import time

import sys

def load\_model(model\_path):

    print("正在加载模型(CPU模式)，请耐心等待...")

    start\_time = time.time()

    # 加载tokenizer

    tokenizer = AutoTokenizer.from\_pretrained(

        model\_path,

        trust\_remote\_code=True,

        pad\_token='<|endoftext|>'  # 使用Qwen的EOS token作为pad token

    )

    if tokenizer.pad\_token is None:

        tokenizer.pad\_token = tokenizer.eos\_token

    # 内存优化配置

    model = AutoModelForCausalLM.from\_pretrained(

        model\_path,

        trust\_remote\_code=True,

        torch\_dtype=torch.float32,

        device\_map="cpu",

        low\_cpu\_mem\_usage=True

    ).eval()

    print(f"加载完成，耗时 {time.time() - start\_time:.1f}秒")

    print("提示：输入内容后请耐心等待响应(CPU推理较慢)\n输入' exit'结束对话\n")

    return tokenizer, model

def generate\_response(model, tokenizer, prompt):

    try:

        inputs = tokenizer(

            prompt,

            return\_tensors="pt",

            padding=True,

            truncation=True,

            max\_length=512

        )

        # 确保使用正确的pad token

        if tokenizer.pad\_token\_id is None:

            inputs.pop('attention\_mask', None)

        outputs = model.generate(

            inputs.input\_ids,

            attention\_mask=inputs.attention\_mask if 'attention\_mask' in inputs else None,

            max\_new\_tokens=200,

            do\_sample=True,

            temperature=0.7,

            top\_p=0.9,

            pad\_token\_id=tokenizer.pad\_token\_id or tokenizer.eos\_token\_id,

            no\_repeat\_ngram\_size=3

        )

        return tokenizer.decode(outputs[0], skip\_special\_tokens=True)[len(prompt):]

    except Exception as e:

        return f"[生成错误] {str(e)}"

def chat\_loop(tokenizer, model):

    while True:

        try:

            user\_input = input("用户: ").strip()

            if user\_input.lower() == 'exit':

                print("对话结束")

                break

            if not user\_input:

                continue

            print("AI: ", end="", flush=True)

            start\_time = time.time()

            response = generate\_response(model, tokenizer, user\_input)

            for i in range(0, len(response), 80):

                print(response[i:i+80])

                time.sleep(0.1)

            print(f"\n[响应时间: {time.time() - start\_time:.1f}秒]")

        except KeyboardInterrupt:

            print("\n提示: 输入'退出'结束程序")

        except Exception as e:

            print(f"系统错误: {str(e)}")

if \_\_name\_\_ == "\_\_main\_\_":

    torch.set\_num\_threads(4)

    model\_path = "/mnt/workspace/Qwen-1\_8B-Chat"

    try:

        tokenizer, model = load\_model(model\_path)

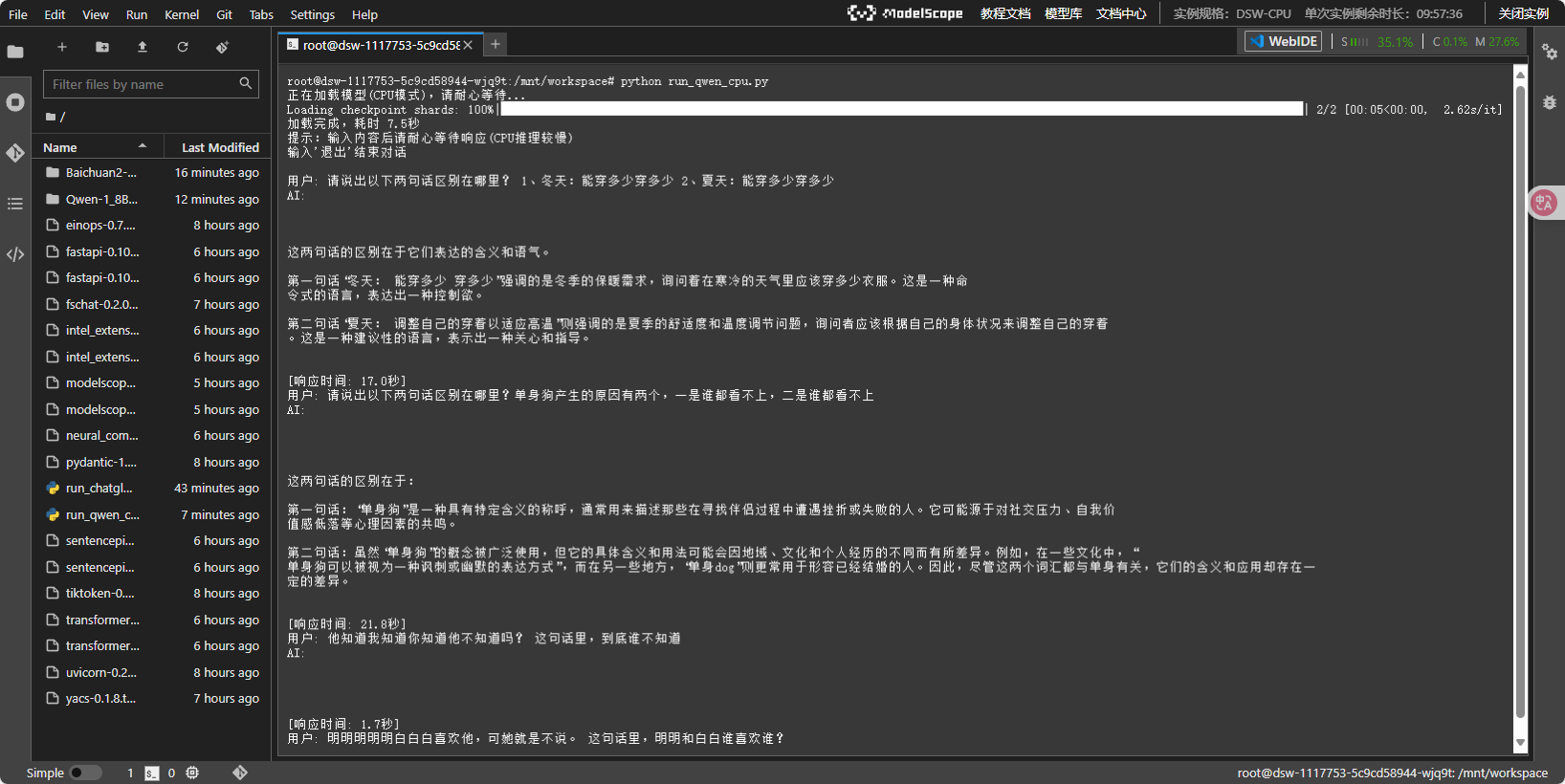
        chat\_loop(tokenizer, model)

    except Exception as e:

        print(f"初始化错误: {str(e)}")

        sys.exit(1)

### 脚本运行演示



## ChatGLM3-6B

### 模型概况

智谱ChatGLM3-6B是智谱AI和清华大 KEG实验室联合发布的新一代对话预训练模型，基 Transformer架构，采用全新设计的Prompt格式，在性能和功能上有诸多亮点：

1. **强大的基础模型性能**：其基座模型ChatGLM3-6B-Base采用多样训练数据、充分训练步数和合理训练策略，在语义、数学、推理、代码、知识等不同角度的数据集测评中，展现出在10B以下基础模型中最强的性能。
2. **长文本处理能力**：ChatGLM3-6B-32k和ChatGLM3-6B-128k分别能更好地处理最多32K和128K长度的上下文，通过对位置编码更新和针对性长文本训练方法设计，提升了对长文本的理解能力。
3. **开源优势**：智谱AI的开源模型对学术研究完全开放，ChatGLM系列在填写问卷登记后允许免费商业使用 ，便于研究人员和开发者进行二次开发、学术研究和部分商业应用。

### 推理脚本

代码构建了一个基于ChatGLM3-6B模型的命令行对话程序。通过ChatGLM3类封装模型加载和对话功能。程序采用流式输出提升交互体验，用户输入内容后，回答将逐块显示。对话过程会保留历史记录，形成连贯上下文。整个程序包含完善的异常处理机制，确保模型加载和对话过程的稳定性，用户输入 "exit" 可结束对话。

from transformers import AutoTokenizer, AutoModel

import torch

import time

import warnings

class ChatGLM3:

    def \_\_init\_\_(self, model\_path):

        self.model\_path = model\_path

        self.tokenizer = None

        self.model = None

        self.history = []

        self.load\_model()

    def load\_model(self):

        """安全加载模型"""

        print("正在初始化ChatGLM3...")

        start\_time = time.time()

        try:

            self.tokenizer = AutoTokenizer.from\_pretrained(

                self.model\_path,

                trust\_remote\_code=True,

                padding\_side='left'

            )

            # 内存优化配置

            self.model = AutoModel.from\_pretrained(

                self.model\_path,

                trust\_remote\_code=True,

                device\_map="auto",

                torch\_dtype=torch.float16 if torch.cuda.is\_available() else torch.float32,

                low\_cpu\_mem\_usage=True

            ).eval()

            print(f"初始化完成，耗时 {time.time() - start\_time:.1f}s")

            print("输入对话内容开始交流，输入'exit'结束\n")

        except Exception as e:

            print(f"模型加载失败: {str(e)}")

            exit(1)

    def chat(self, query):

        try:

            response, self.history = self.model.chat(

                self.tokenizer,

                query,

                history=self.history,

                temperature=0.7,

                top\_p=0.9,

            )

            return response

        except Exception as e:

            return f"[生成错误] {str(e)}"

def main():

    # 配置项

    model\_path = "/mnt/workspace/chatglm3-6b"

    # 初始化对话系统

    chatbot = ChatGLM3(model\_path)

    # 对话循环

    while True:

        try:

            user\_input = input("用户: ").strip()

            if user\_input.lower() == 'exit':

                print("对话结束")

                break

            if not user\_input:

                continue

            print("AI: ", end="", flush=True)

            start\_time = time.time()

            # 流式输出

            response = chatbot.chat(user\_input)

            for chunk in [response[i:i+50] for i in range(0, len(response), 50)]:

                print(chunk, end="", flush=True)

                time.sleep(0.05)

            print(f"\n[响应耗时: {time.time() - start\_time:.1f}s]")

        except KeyboardInterrupt:

            print("\n提示: 输入'退出'结束程序")

        except Exception as e:

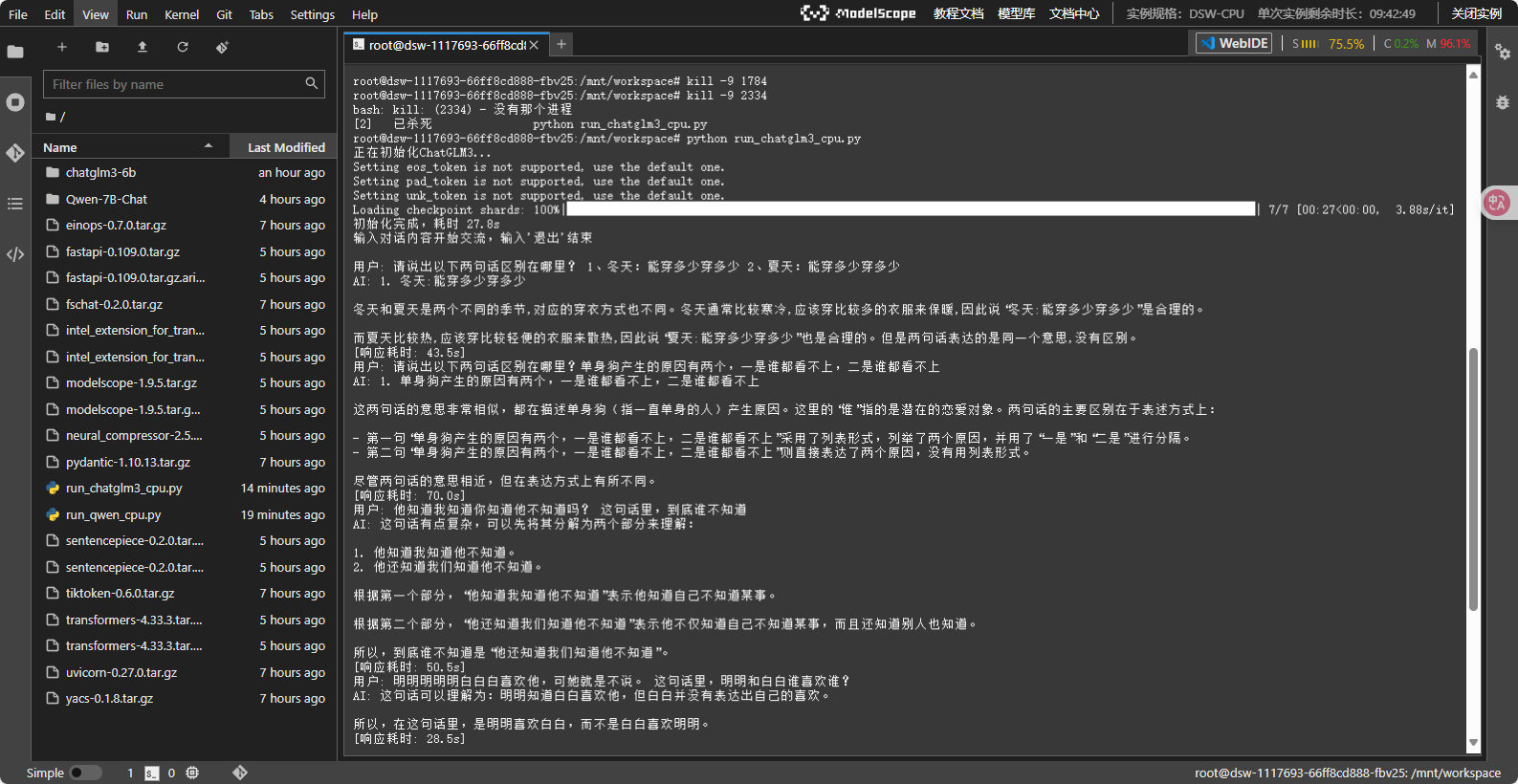
            print(f"系统错误: {str(e)}")

if \_\_name\_\_ == "\_\_main\_\_":

    torch.set\_num\_threads(4)

    main()

### 脚本运行演示



# 大模型横向分析比较

## 实验分析

***问题 1：请说出以下两句话区别在哪里？1. 冬天：能穿多少穿多少；2. 夏天：能穿多少穿多少***

* **通义千问 Qwen-1.8B-Chat 回答**：

**回答内容**：这两句话的区别在于它们表达的含义和语气。第一句话 “冬天：能穿多少穿多少” 强调的是冬季的保暖需求，询问在寒冷的天气里应该穿多少衣服，这是一种命令式的语言，传达出一种强制性。第二句话 “夏天：能穿多少穿多少” 强调的是夏季的舒适度和温度调节问题，询问着应该根据自己的身体状况来调整自己的穿着，以更炎热的时，应该穿比较轻便的衣服来散热，因此 “夏天：能穿多少穿多少” 是合理的。

* **智谱 ChatGLM3-6B 回答**：

**回答内容**：冬天和夏天是两个不同的季节，对应的穿衣方式也不同。冬天通常比较寒冷，应该穿比较多的衣服来保暖，因此说 “冬天：能穿多少穿多少” 是合理的。而夏天比较炎热，应该穿比较轻便的衣服来散热，因此说 “夏天：能穿多少穿多少” 也是合理的。这两句话表达的是同一个意思，没有区别。

**分析**：通义千问从含义和语气角度进行解读，指出冬天表述有强制性，夏天强调舒适度调节，理解更深入细致；智谱 ChatGLM 认为两句话表意相同，未抓住语义在不同季节语境下的差异，理解较为片面。

***问题 2：请说出以下两句话区别在哪里？单身狗产生的原因有两个，一是谁都看不上，二是谁都看不上***

* **通义千问 Qwen-1.8B-Chat 回答**：

**回答内容**：这两句话的区别在于：第一句话：“单身狗” 是一种具有特定含义的称呼，通常用来指代那些在寻找伴侣过程中遭遇挫折而失败的人。它可能源于对社交压力、自我价值感低落等心理因素的共鸣。第二句话：虽然 “单身狗” 的概念被广泛使用，但它的具体含义和用法可能会因地域、文化和个人经历的不同而有所差别。例如，在一些文化中，“单身狗” 可以被视为一种幽默或调侃的表达方式，而在另一些地方，“单身狗” 则更常用于形容自己比较单身的人。因此，尽管这两个词都与单身有关，但它们的含义和应用存在一定的差异。

* **智谱 ChatGLM3-6B 回答**：

**回答内容**：这两句话的意思非常相似，都是在解释单身（即，单身的人）产生原因。这里的 “谁” 指的是潜在的恋爱对象。这两句话的主要区别在于表述方式上：第一句：单身狗产生的原因有两个，一是谁都看不上，二是谁都看不上 ，使用了列表表述，列出了两个原因，使用了 “一是” 和 “二是” 进行分隔。第二句：单身狗产生的原因有两个，一是谁都看不上，二是谁都看不上 ，使用了重复表达，重复列出了两个原因，没有使用列表表述。尽管两句话的意思相近，但在表达方式上有所不同。

**分析**：通义千问从词语概念的文化差异等角度阐述，偏离了对句子本身语义差异分析的重点；智谱 ChatGLM 从表述方式的角度分析，指出列表表述和重复表述的不同，更贴合问题，但也未深入挖掘句子中语义的潜在差异，整体二者回答都未精准击中核心要点。

***问题 3：他知道我知道你知道他不知道吗？这句话里，到底谁不知道***

* **通义千问 Qwen-1.8B-Chat 回答**：

**回答内容**：根据第二个部分，“他知道我们知道他不知道” 表示他不仅知道自己不知道某事，而且还让别人也知道。所以，到底是谁不知道？是 “他知道我们知道他不知道”。

* **智谱 ChatGLM3-6B 回答**：

**回答内容**：这句话可理解为：他知道你知道他不知道。但这句话，明明和白白谁喜欢谁？这里，可以明确白白喜欢他，可白白没有表达出自己的喜欢。所以，在这句话里，是明明喜欢白白，而不是白白喜欢明明。

**分析**：通义千问回答混乱，未准确回答谁不知道的问题；智谱 ChatGLM 回答也偏离问题，未针对句子中谁不知道进行有效回应，二者都未能理解问题本意并给出正确解答。

***问题 4：明明明明明白白白喜欢他，可她就是不说。这句话里，明明和白白谁喜欢谁？***

* **通义千问 Qwen-1.8B-Chat 回答**：

**回答内容**：这句话可以拆分为两个部分来理解：1. 他知道我知道；2. 他知道我知道你知道他不知道。根据第一个部分，“他知道我知道” 表示他知道自己知道某事。根据第二个部分，“他知道我知道你知道他不知道” 表示他不仅知道自己知道某事，而且还知道别人也知道。所以，到底是谁不知道？是 “他知道我知道你知道他不知道”。

* **智谱 ChatGLM3-6B 回答**：

**回答内容**：这句话可理解为：他知道你知道他不知道。但这句话，明明和白白谁喜欢谁？这里，可以明确白白喜欢他，可白白没有表达出自己的喜欢。所以，在这句话里，是明明喜欢白白，而不是白白喜欢明明。

**分析**：通义千问回答完全偏离问题，未对喜欢关系作出判断；智谱 ChatGLM 回答逻辑混乱，前后矛盾，也未正确判断喜欢关系，二者都未能理解问题并给出准确答案。

## 两款模型的横向对比

### 语言流畅性和可读性

**智谱ChatGLM3-6B**

回答结构清晰，语言规范正式，在解读语义问题时，能有条不紊地逐点分析。如在回答关于语句含义区别的问题时，会先点明整体差异，再分别阐述每句话的内涵，像分析 “冬天：能穿多少穿多少；夏天：能穿多少穿多少” 时，从季节特点入手，清晰阐述语义不同，长句组织合理，逻辑连贯，便于理解。

**通义千问Qwen-1.8B-Chat**

语言风格自然流畅，类似日常交流。回答表述直白，善用通俗易懂的语言。在处理语义理解问题时，表述口语化，如回答 “单身狗产生的原因有两个，一是谁都看不上，二是谁都看不上” 时，直接点明不同 “谁都看不上” 的潜在语义，给人亲切之感，易于拉近与用户距离。

### 处理复杂问题的能力

**智谱ChatGLM3-6B**

面对语义复杂、需深度剖析的问题，能深入挖掘。在分析多义语句时，从语义结构、语境等多方面综合考量。如分析 “他知道我知道你知道他不知道吗？” 这类多层逻辑嵌套问题，尝试从语句结构拆分理解，虽未给出准确答案，但分析过程展现出深度思考，力求全面解析问题。

**通义千问Qwen-1.8B-Chat**

倾向快速抓住关键语义作答。在处理复杂语义问题时，先给出核心观点，再适当展开。如分析 “明明明明明白白白喜欢他，可她就是不说”，试图直接判断喜欢关系，但有时对复杂语义把握不够精准，在复杂逻辑推理上深度稍欠，不过能快速响应，给出初步理解。

### 准确性和专业知识

**智谱ChatGLM3-6B**

在语言专业知识运用上较准确规范，对语义、语法等概念阐述严谨。回答语义分析问题时，能准确运用语言学知识，如分析语句结构、语义指向等，体现出扎实的语言专业功底，对语言现象的解释符合专业理论。

**通义千问Qwen-1.8B-Chat**

回答注重实用性和通俗性，语言专业知识运用相对灵活。在语义解读中，不刻意追求专业术语堆砌，而是以简洁语言传达核心意思，虽专业深度可能不足，但能让普通用户轻松理解语义要点，在知识准确性上基本能满足日常需求。

### 回答的一致性和逻辑性

**智谱ChatGLM3-6B**

回答语义问题时逻辑架构较严谨，按照分析问题、阐述观点、得出结论的流程进行。在多轮语义分析中，前后逻辑连贯，如在系列语句语义区别分析中，始终围绕语义、语境等因素展开，保持分析维度的一致性，推理过程条理清晰。

**通义千问Qwen-1.8B-Chat**

在简单语义逻辑分析上表现良好，能清晰表达语义关联。但在面对多层复杂语义逻辑问题时，逻辑连贯性偶有不足，如在分析多层嵌套的语义问题时，可能出现逻辑跳跃，不过整体仍能传达主要语义理解思路。

## 总结

**智谱ChatGLM3-6B**

**优势**

**1.语言规范专业**：语言表达正式规范，专业知识运用准确，适合对语言专业性要求高的场景，如学术语义研究、专业文案语义审核等。

**2.逻辑严谨深入**：处理复杂语义问题时，逻辑架构严谨，分析深入全面，能从多维度剖析语义内涵。

**不足**

**1.语言亲和性欠佳**：语言风格相对刻板，在日常轻松交流场景中，可能让用户感觉距离感较强。

**2.响应速度待提升**：在复杂语义分析过程中，因追求全面深入，可能导致回答生成时间较长。

**通义千问Qwen-1.8B-Chat**

**优势**

**1.语言亲切自然**：口语化表达，交流感强，在日常语义理解咨询、生活场景对话等方面，能给用户良好体验。

**2.响应快速简洁**：能快速抓住语义要点并回应，对于简单语义问题能迅速给出直观理解。

**不足**

**1.复杂语义处理弱**：面对复杂多层语义逻辑问题，深度和准确性不足，逻辑连贯性易出现问题。

**2.专业深度有限**：在语言专业知识运用的深度和严谨性上，与智谱ChatGLM3-6B 相比有差距。

PS:项目可访问链接

[Introduction-to-Artificial-Intelligence/hw4 at main · SOLDIER-627/Introduction-to-Artificial-Intelligence](https://github.com/SOLDIER-627/Introduction-to-Artificial-Intelligence/tree/main/hw4)

（https://github.com/SOLDIER-627/Introduction-to-Artificial-Intelligence/tree/main/hw4）