数据工程相关内容

2. **数据工程篇**

数据工程在构建稳定的数据流转链路方面起着关键作用，其涵盖了从原始文本数据到符合模型输入格式数据集的一系列处理流程。首先，数据工程的整体目标是构建稳定的数据流转链路，从原始文本数据逐步转化为清洗、标注、审核后的语料，最终形成符合模型输入格式的数据集。同时要保证数据可用性、完整性、质量以及结构统一性，并且支持训练、微调、推理RAG、评估测试等多阶段使用场景。数据来源主要有甲方提供和自主收集两种。甲方提供是常见形式，如客户工单、业务知识库等，需尽早明确数据格式、内容范围、隐私边界等，沟通重点在于明确数据目标用途、是否可公开训练及是否需要脱敏。自主收集虽成本高、周期长，但更具通用性和自主性，收集方式包括手动采集、网页爬虫、调用开放接口以及AI生成数据等。以微调训练金融科技领域问答模型为例，AI生成数据的流程为初始化，设置小白和资深两种对话风格配置，每种风格包含系统角色设定、示例对话、温度参数，通过数据生成循环构建对话上下文，随机选取问题调用AI接口获取风格化回答并结构化存储，加入延迟防止API限流，最后合并数据、处理异常情况并保存为格式化JSON文件。原始数据往往杂乱无章，因此需要进行系统的清洗与预处理。内容清洗方面，要去除HTML标签、表情等无效信息，统一编码格式并规范文本结构。分段处理需按语义或结构分段，确保模型输入粒度适当。内容筛选要删除空白、重复及无意义内容，过滤违法违规和广告类内容。根据用途，要对数据进行结构化和标注。标注方式有自动标注（适合结构化文本等）、人工标注（适用于复杂任务）、AI辅助 + 人工校对（效率与质量平衡）。数据集结构标准化需根据目标任务设计，如问答为{"question": "", "answer": ""}等形式，统一存储格式有JSONL、CSV、Parquet。为保证数据质量，避免模型效果被低质样本污染，需要进行数据审查与质检。审查内容包括是否存在空文本、文本长度是否合理等。常规校验机制有自动规则校验、随机抽样人工审核、自动评分 + 低分样本再审。为消除重复文本，避免模型过拟合，要进行文本去重与相似性筛查。实现思路是使用Embedding模型将文本转为向量表示，计算文本对之间的相似度，设置合理相似度阈值，保留更优版本文本，剔除相似冗余样本。在数据处理过程中，要记录样本数量、保留比例等数据统计信息，并建立数据集版本控制机制，便于溯源、复现和对比实验效果。尤其是面向企业应用时，必须落实数据安全要求，进行敏感字段脱敏、权限分级、明确数据使用范围，并加入审计与日志记录。以下为相关示例代码：```pythonimport jsonimport timeimport randomfrom zhipuai import ZhipuAI# 初始化模型client = ZhipuAI(api\_key="48b393b3474e43f3b7eafceb662db81a.xxxxxx")style\_config = { "小白": { "system\_prompt": "你是一位区块链科普专家，需要用最简单易懂的方式向完全不了解技术的普通人解释区块链概念。要求：\n1. 使用生活化比喻（如'像大家一起记账本'）\n2. 避免专业术语，必须使用时立即解释\n3. 回答控制在3句话以内\n4. 可适当使用🌐⛓️💡等表情符号", "examples": [ {"role": "user", "content": "区块链到底是什么？"}, {"role": "assistant", "content": "想象一群人在共同维护一个公开账本🌐，每笔交易都记录在所有人的本子上，想作弊就得改掉大部分人的记录，这几乎不可能⛓️！"}, {"role": "user", "content": "比特币为什么值钱？"}, {"role": "assistant", "content": "就像限量版球星卡💎，总量固定(2100万枚)，越来越多人想要时价格就上涨啦！"} ], "temperature": 0.2 # 低温度值保证回答稳定性 }, "资深": { "system\_prompt": "你是区块链核心技术研究员，需满足：\n1. 使用专业术语（如'椭圆曲线数字签名''梅克尔帕特里夏树'）\n2. 包含技术实现细节\n3. 必要时给出公式/代码片段\n4. 引用最新论文成果（如2023 IEEE S&P会议）", "examples": [ {"role": "user", "content": "请解释zk - SNARKs的工作原理"}, {"role": "assistant", "content": "zk - SNARKs依赖QAP(二次算术程序)将计算转化为多项式关系：\n1) Setup生成CRS(公共参考串)\n2) Prover构造π=(A,B,C)满足A\*B=C+H\*Z\n3) Verifier通过双线性配对验证e(A,B)=e(C,g)⋅e(H,g^z)\n参考：Groth16方案优化后验证仅需3个配对运算。"}, {"role": "user", "content": "分析EIP - 4844对L2的影响"}, {"role": "assistant", "content": "Proto - danksharding通过引入blob携带交易(每个blob≈125KB)，使Rollups数据可用性成本降低10 - 100倍。实测显示：\n- Optimism交易费从$0.3→$0.007\n- zkSync证明生成时间减少40%"} ], "temperature": 0.5 # 中等温度值允许一定创造性 }}def generate\_style\_data(style\_name, num\_samples = 50): config = style\_config[style\_name] data = [] # 构建对话上下文 = 系统提示 + 示例对话 messages = [ {"role": "system", "content": config["system\_prompt"]}, \*config["examples"] ] # 预定义的用户问题集（可扩展） user\_inputs = [ "比特币和区块链是什么关系？", "智能合约真的安全吗？", "普通人怎么参与DeFi？", "解释一下零知识证明", "为什么NFT能卖那么贵？", "区块链能防止数据造假吗？" ] for \_ in range(num\_samples): try: # 随机选择问题 user\_msg = random.choice(user\_inputs) # 当前对话 = 历史上下文 + 新问题 current\_messages = messages + [ {"role": "user", "content": user\_msg} ] # 调用API（修正模型名称） response = client.chat.completions.create( model = "glm - 4 - plus", messages = current\_messages, temperature = config["temperature"], max\_tokens = 150 ) # 获取回复内容 reply = response.choices[0].message.content data.append({ "user": user\_msg, "assistant": reply, "style": style\_name }) print("获取最新回复内容: ", data[-1]) time.sleep(0.5) # 限流 except Exception as e: print(f"生成失败：{str(e)}") return dataif \_\_name\_\_ == '\_\_main\_\_': all\_data = [] try: print("开始生成小白风格数据...") gentle\_data = generate\_style\_data("小白", 10) all\_data.extend(gentle\_data) print("开始生成资深风格数据...") gentle\_data = generate\_style\_data("资深", 10) all\_data.extend(gentle\_data) except KeyboardInterrupt: print("\n用户中断，保存已生成数据...") finally: with open("style\_chat\_data.json", "w", encoding = "utf - 8") as f: json.dump(all\_data, f, ensure\_ascii = False, indent = 2) print(f"数据已保存，有效样本数：{len(all\_data)}")``````pythonimport jsonfrom typing import List, Dictdef is\_meaningless(text: str) -> bool: """检查是否是无意义内容""" # 空白或超短内容 if len(text.strip()) < 5: return True # 常见无意义模式 meaningless\_phrases = [ "我不知道", "无法回答", "这个问题", "请重新提问" ] return any(phrase in text for phrase in meaningless\_phrases)def simple\_clean(data: List[Dict]) -> List[Dict]: """基础清洗流程""" seen = set() # 用于去重 cleaned\_data = [] for item in data: text = item["assistant"].strip() # 检查空白/重复/无意义 if (not text or text in seen or is\_meaningless(text)): continue seen.add(text) cleaned\_data.append({ "user": item["user"], "assistant": text, "style": item["style"] }) return cleaned\_dataif \_\_name\_\_ == '\_\_main\_\_': # 加载原始数据 with open('style\_chat\_data.json', 'r', encoding='utf - 8') as f: raw\_data = json.load(f) # 执行清洗 cleaned\_data = simple\_clean(raw\_data) # 保存结果 with open('cleaned\_data.json', 'w', encoding='utf - 8') as f: json.dump(cleaned\_data, f, ensure\_ascii = False, indent = 2) # 打印简单报告 print(f"原始数据量: {len(raw\_data)}") print(f"清洗后保留: {len(cleaned\_data)}") print(f"过滤数量: {len(raw\_data) - len(cleaned\_data)}")``````pythonimport jsonfrom sentence\_transformers import SentenceTransformerimport numpy as npdef deduplicate\_data(data\_list, threshold = 0.9): """ 语义去重：对 assistant 回复去重 :param data\_list: List[dict], 每个 dict 包含 user / assistant / style :param threshold: float, 相似度阈值 :return: 去重后的 List[dict] """ model = SentenceTransformer("/root/autodl - tmp/models/thomas/text2vec - base - chinese") replies = [item["assistant"] for item in data\_list] embeddings = model.encode(replies) kept = [] seen = set() for i, emb\_i in enumerate(embeddings): if i in seen: continue keep = True for j, emb\_j in enumerate(embeddings): if j == i or j in seen: continue sim = np.dot(emb\_i, emb\_j) / (np.linalg.norm(emb\_i) \* np.linalg.norm(emb\_j)) print(f"sim({i}, {j}) = {sim:.4f}") if sim > threshold: seen.add(j) kept.append(data\_list[i]) return keptif \_\_name\_\_ == '\_\_main\_\_': with open("style\_chat\_data.json", "r", encoding = "utf - 8") as f: all\_data = json.loads(f.read()) cleaned\_data = deduplicate\_data(all\_data, 0.5)```

2.1 **数据工程的整体目标**

* 构建稳定的数据流转链路，从**原始文本数据** → **清洗/标注/审核后的语料** → **符合模型输入格式的数据集**
* 保证数据**可用性、完整性、质量、结构统一性**
* 支持多阶段使用场景：**训练/微调、推理RAG、评估测试等**

2.2 **数据来源**

2.2.1 **甲方提供**

* 最常见的形式，如客户工单、业务知识库、客服聊天记录、FAQ 等。
* 需尽早明确数据格式、内容范围、隐私边界。
* **沟通重点**：明确数据目标用途、是否可公开训练、是否需要脱敏。

2.2.2 **自主收集**

* 成本高、周期长，但更具通用性和自主性。
* 数据收集方式：
* **手动采集**（如论坛、PDF、博客文章等）
* **网页爬虫**（需遵循 robots 协议与合法边界）
* **调用开放接口**（如新闻、百科类 API）
* **AI 生成数据**（如大模型生成问答对，用于训练增强）

2.2.2.1 **AI 生成数据示例**

以微调训练金融科技领域问答模型为例，通过 AI 生成训练数据。

**流程：**

1. **初始化**

* 设置两种对话风格配置：
* 小白风格：通俗易懂的比喻解释，限制术语使用
* 资深风格：专业术语和技术细节，要求严谨准确
* 每种风格包含：
* 系统角色设定（system\_prompt）
* 示例对话（examples）
* 温度参数（控制回答随机性）

1. **数据生成循环**

* 构建包含系统提示和示例的对话上下文
* 从预设问题列表中随机选取问题
* 调用 AI 接口获取风格化回答
* 结构化存储问答对（问题、回答、风格标签）
* 加入延迟防止 API 限流

1. **结果处理**

* 合并所有生成的问答数据
* 处理可能的异常情况（如中断请求）
* 将最终数据集保存为格式化的JSON文件

**示例代码：**

|  |
| --- |
| Python import json import time import random from zhipuai import ZhipuAI  # 初始化模型 client = ZhipuAI(api\_key="48b393b3474e43f3b7eafceb662db81a.xxxxxx")  style\_config = {  "小白": {  "system\_prompt": "你是一位区块链科普专家，需要用最简单易懂的方式向完全不了解技术的普通人解释区块链概念。要求：\n1. 使用生活化比喻（如'像大家一起记账本'）\n2. 避免专业术语，必须使用时立即解释\n3. 回答控制在3句话以内\n4. 可适当使用🌐⛓️💡等表情符号",  "examples": [  {"role": "user", "content": "区块链到底是什么？"},  {"role": "assistant",  "content": "想象一群人在共同维护一个公开账本🌐，每笔交易都记录在所有人的本子上，想作弊就得改掉大部分人的记录，这几乎不可能⛓️！"},  {"role": "user", "content": "比特币为什么值钱？"},  {"role": "assistant", "content": "就像限量版球星卡💎，总量固定(2100万枚)，越来越多人想要时价格就上涨啦！"}  ],  "temperature": 0.2 # 低温度值保证回答稳定性  },  "资深": {  "system\_prompt": "你是区块链核心技术研究员，需满足：\n1. 使用专业术语（如'椭圆曲线数字签名''梅克尔帕特里夏树'）\n2. 包含技术实现细节\n3. 必要时给出公式/代码片段\n4. 引用最新论文成果（如2023 IEEE S&P会议）",  "examples": [  {"role": "user", "content": "请解释zk-SNARKs的工作原理"},  {"role": "assistant",  "content": "zk-SNARKs依赖QAP(二次算术程序)将计算转化为多项式关系：\n1) Setup生成CRS(公共参考串)\n2) Prover构造π=(A,B,C)满足A\*B=C+H\*Z\n3) Verifier通过双线性配对验证e(A,B)=e(C,g)⋅e(H,g^z)\n参考：Groth16方案优化后验证仅需3个配对运算。"},  {"role": "user", "content": "分析EIP-4844对L2的影响"},  {"role": "assistant",  "content": "Proto-danksharding通过引入blob携带交易(每个blob≈125KB)，使Rollups数据可用性成本降低10-100倍。实测显示：\n- Optimism交易费从$0.3→$0.007\n- zkSync证明生成时间减少40%"}  ],  "temperature": 0.5 # 中等温度值允许一定创造性  } }   def generate\_style\_data(style\_name, num\_samples=50):  config = style\_config[style\_name]  data = []   # 构建对话上下文 = 系统提示 + 示例对话  messages = [  {"role": "system", "content": config["system\_prompt"]},  \*config["examples"]  ]   # 预定义的用户问题集（可扩展）  user\_inputs = [  "比特币和区块链是什么关系？",  "智能合约真的安全吗？",  "普通人怎么参与DeFi？",  "解释一下零知识证明",  "为什么NFT能卖那么贵？",  "区块链能防止数据造假吗？"  ]   for \_ in range(num\_samples):  try:  # 随机选择问题  user\_msg = random.choice(user\_inputs)   # 当前对话 = 历史上下文 + 新问题  current\_messages = messages + [  {"role": "user", "content": user\_msg}  ]   # 调用API（修正模型名称）  response = client.chat.completions.create(  model="glm-4-plus",  messages=current\_messages,  temperature=config["temperature"],  max\_tokens=150  )   # 获取回复内容  reply = response.choices[0].message.content  data.append({  "user": user\_msg,  "assistant": reply,  "style": style\_name  })  print("获取最新回复内容: ", data[-1])  time.sleep(0.5) # 限流   except Exception as e:  print(f"生成失败：{str(e)}")  return data   if \_\_name\_\_ == '\_\_main\_\_':  all\_data = []   try:  print("开始生成小白风格数据...")  gentle\_data = generate\_style\_data("小白", 10)  all\_data.extend(gentle\_data)   print("开始生成资深风格数据...")  gentle\_data = generate\_style\_data("资深", 10)  all\_data.extend(gentle\_data)   except KeyboardInterrupt:  print("\n用户中断，保存已生成数据...")  finally:  with open("style\_chat\_data.json", "w", encoding="utf-8") as f:  json.dump(all\_data, f, ensure\_ascii=False, indent=2)  print(f"数据已保存，有效样本数：{len(all\_data)}") |

2.3 **数据清洗与预处理**

原始数据往往杂乱无章，需系统清洗处理。

* **内容清洗**
* 去除 HTML 标签、表情、特殊符号、水印等无效信息
* 统一编码格式，规范文本结构（如换行、缩进）
* **分段处理**
* 按语义或结构分段，确保模型输入粒度适当
* 避免文本过长或过短，提升模型效果
* **内容筛选**
* 删除空白内容、重复内容、无意义语段

|  |
| --- |
| Python import json from typing import List, Dict   def is\_meaningless(text: str) -> bool:  *"""检查是否是无意义内容"""* # 空白或超短内容  if len(text.strip()) < 5:  return True   # 常见无意义模式  meaningless\_phrases = [  "我不知道", "无法回答",  "这个问题", "请重新提问"  ]  return any(phrase in text for phrase in meaningless\_phrases)   def simple\_clean(data: List[Dict]) -> List[Dict]:  *"""基础清洗流程"""* seen = set() # 用于去重  cleaned\_data = []   for item in data:  text = item["assistant"].strip()   # 检查空白/重复/无意义  if (not text or  text in seen or  is\_meaningless(text)):  continue   seen.add(text)  cleaned\_data.append({  "user": item["user"],  "assistant": text,  "style": item["style"]  })   return cleaned\_data   if \_\_name\_\_ == '\_\_main\_\_':  # 加载原始数据  with open('style\_chat\_data.json', 'r', encoding='utf-8') as f:  raw\_data = json.load(f)   # 执行清洗  cleaned\_data = simple\_clean(raw\_data)   # 保存结果  with open('cleaned\_data.json', 'w', encoding='utf-8') as f:  json.dump(cleaned\_data, f, ensure\_ascii=False, indent=2)   # 打印简单报告  print(f"原始数据量: {len(raw\_data)}")  print(f"清洗后保留: {len(cleaned\_data)}")  print(f"过滤数量: {len(raw\_data) - len(cleaned\_data)}") |

* 过滤掉违法违规、广告类内容，保障数据合规

2.4 **数据标注与格式化**

根据用途，对数据进行结构化和标注。

* **标注方式**
* 自动标注：适合结构化文本、已有知识图谱辅助生成
* 人工标注：适用于对话意图、情感分类、多轮对话标记等复杂任务
* AI 辅助+人工校对：效率与质量的平衡方式
* **数据集结构标准化**
* 根据目标任务结构设计
* 问答：{"question": "", "answer": ""}
* 多轮对话：{"dialog": [{"role": "user", "content": ""}, {"role": "assistant", "content": ""}, ...]}
* 分类任务：{"text": "", "label": ""}
* **统一存储格式**
* JSONL
* CSV
* Parquet

2.5 **数据审查与质检**

保证数据质量，避免模型效果被低质样本污染。

* **审查内容**
* 是否存在空文本？
* 文本长度是否合理？
* 是否出现乱码、非法字符？
* 是否有标签缺失、格式错误？
* **常规校验机制**
* 自动规则校验（正则、长度范围）
* 随机抽样人工审核
* 自动评分 + 低分样本再审

2.6 **文本去重与相似性筛查**

消除重复文本，避免模型**过拟合**。

**实现思路：**

1. 使用 **Embedding 模型，**将文本转为向量表示
2. 计算文本对之间的相似度（如**余弦相似度**或**欧氏距离**）
3. 设置合理**相似度阈值**（如 > 0.95 即视为重复）
4. 保留更优版本文本，剔除相似冗余样本

|  |
| --- |
| Python import json from sentence\_transformers import SentenceTransformer import numpy as np   def deduplicate\_data(data\_list, threshold=0.9):  *"""*  *语义去重：对 assistant 回复去重*  *:param data\_list: List[dict], 每个 dict 包含 user / assistant / style*  *:param threshold: float, 相似度阈值*  *:return: 去重后的 List[dict]*  *"""* model = SentenceTransformer("/root/autodl-tmp/models/thomas/text2vec-base-chinese")   replies = [item["assistant"] for item in data\_list]  embeddings = model.encode(replies)  kept = []  seen = set()   for i, emb\_i in enumerate(embeddings):  if i in seen:  continue   keep = True  for j, emb\_j in enumerate(embeddings):  if j == i or j in seen:  continue   sim = np.dot(emb\_i, emb\_j) / (np.linalg.norm(emb\_i) \* np.linalg.norm(emb\_j))  print(f"sim({i}, {j}) = {sim:.4f}")  if sim > threshold:  seen.add(j)  kept.append(data\_list[i])   return kept   if \_\_name\_\_ == '\_\_main\_\_':  with open("style\_chat\_data.json", "r", encoding="utf-8") as f:  all\_data = json.loads(f.read())  cleaned\_data = deduplicate\_data(all\_data, 0.5) |

2.7 **数据统计与版本控制**

* 记录每轮数据处理中的：样本数量、保留比例、清洗率、标注覆盖率等
* 建立数据集版本控制机制（如 v1、v2、实验版、正式版）
* 便于溯源、复现、对比实验效果

2.8 **数据安全与隐私保护**

尤其是面向企业应用时，必须落实数据安全要求。

* 敏感字段脱敏（如姓名、电话、身份证、公司信息等）
* 权限分级，控制访问范围
* 明确数据使用范围（训练 or 部署 or 产品化）
* 加入审计与日志记录