

# Lab-05 | 提示词工程 & 参数高效微调

## 一、实验目标

利用大语言模型，结合提示词工程、参数高效微调等方法抽取新闻文本中的关键事件要素。

## 二、实验内容

**主要目标：**利用大语言模型，结合提示词工程、参数高效微调等方法抽取新闻文本中的关键事件要素。

**数据准备：**自行收集 50 篇新闻报道作为测试数据，主要关于我国产业、政治、科技、经济领域所面临的风险挑战。

- 相关事件类型包括但不限于是：技术封锁、经济制裁、产业安全、贸易摩擦等，尽量提升数据丰富程度。
- 建议针对不同的事件，为每个事件都搜集若干相关报道。例如美国大选、俄乌战争、GPU禁售、对华加征关税四个事件各收集十几篇相关报道（只是举个例子，不要局限于这几个事件）。
- 尽量新颖，尽量提升数据丰富程度。

**额外备注：**可以自己手动收集新闻，也可以自己写一个爬虫自动爬取，也可以调用一些搜索的 api 工具，例如 SearXNG、智谱的 websearch 等等

**实践任务：**利用大模型尝试从新闻文本中抽取以下关键要素：时间，地点，行业，事件相关国家，事件相关主体、客体及其行为，涉及政策/法案/条款内容，事件情感倾向（数值：-5（极负）到 +5（极正）），事件影响范围（分类：社区级/城市级/省级/国家级/世界级），可能导致的未来影响，以及其他更细粒度的事件要素（可自行拓展设计事件要素模板）。

**实验要求：**从以下四种难度递增的角度选择某种或某几种完成实验

1. 使用网页端大模型（如：文心一言、ChatGPT等），并设计提示词；
2. 通过api调用闭源大模型（参考附录部分），并设计提示词；
3. 本地部署开源大模型（如：Qwen系列、LLaMA系列等），并设计提示词；
4. 以参数高效的方式微调开源大模型，构建在事件要素抽取任务上效果更优的模型。

## 三、实验原理

### 1. 提示词工程 (Prompting) <https://arxiv.org/abs/2404.01077>

#### 1.1. 预训练、提示和预测

**提示词工程**（Prompting）是一种通过设计输入提示（prompt）来引导预训练语言模型（PLM）生成目标输出的技术。其核心思想是利用预训练模型在大规模语料中习得的语言模式和知识，通过精心构造的输入上下文（即提示）激活模型内部的相关表示，从而实现特定任务的推理或生成。

在**预训练阶段**，模型通过自监督学习（如掩码语言建模或因果语言建模）在海量文本上学习通用的语言表示，形成对词汇、语法、语义及常识的隐式编码。例如，BERT 或 GPT 系列模型在此阶段学习到丰富的上下文敏感表示。**提示阶段**的关键在于将下游任务转化为模型可理解的输入形式。例如，将文本分类任务转化为“这个电影很\_\_\_\_”的填空形式，模型需根据上下文预测“好”或“差”。**预测阶段**则依赖模型的生成能力，基于提示的语义线索输出符合任务要求的结果。

提示词工程的优势在于无需调整模型参数，仅通过优化提示设计即可实现任务适配。研究表明，合理的提示设计（如添加示例或任务描述）可显著提升模型性能，甚至在某些任务上接近微调的效果。此外，该方法降低了计算成本，适用于资源受限场景（如边缘设备或低算力环境）。

## 1.2. 硬提示与软提示

提示词工程可分为硬提示（Hard Prompting）与软提示（Soft Prompting）两类，二者在设计方式和优化目标上存在显著差异。**硬提示**指直接在输入中嵌入显式文本指令或关键词，通过语义关联引导模型输出，例如在问答任务中，硬提示可能包含任务描述或示例。其优势在于直观且易于调试，但依赖人工设计，难以覆盖复杂任务的多样性。此外，硬提示的性能高度依赖于提示与模型预训练语料的匹配度，若模型未见过类似表述，效果可能受限。

**软提示**则通过学习可训练的嵌入向量（prompt embedding）间接调整模型行为，而非直接修改输入文本。典型方法如**前缀调优**（Prefix Tuning）或**适配器**（Adapters），通过引入少量参数（通常远小于模型总参数量）与原始输入拼接，引导模型生成目标输出。例如，软提示可将“电影评论”任务映射为一组隐式向量，动态激活模型中与情感分析相关的表示。相比硬提示，软提示更灵活且能适应多跳推理或跨模态任务，但需额外训练提示向量，增加了计算开销。

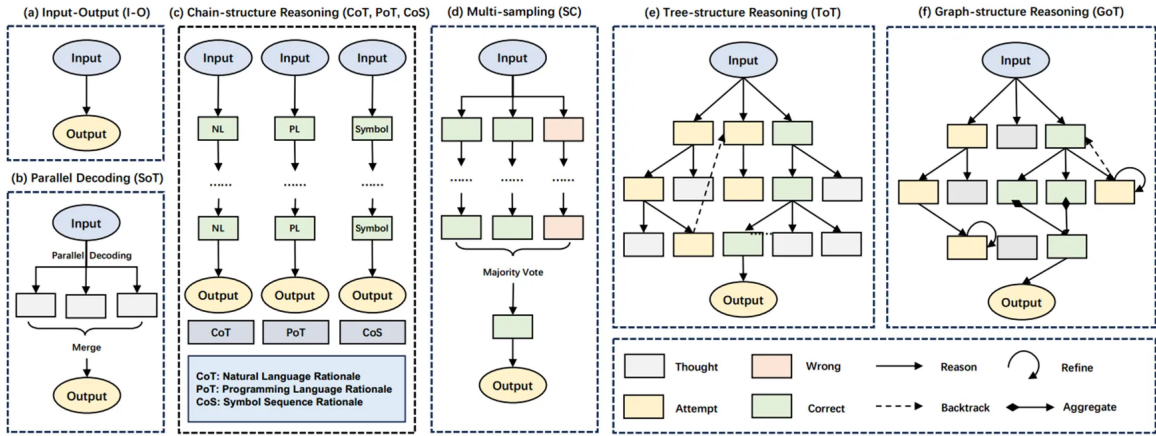
近年来，研究者提出混合策略（如**硬-软结合提示**），结合两者的优点。例如，在硬提示中嵌入可学习的软提示向量，既保留人工设计的可控性，又增强模型对任务的适应性。这类方法在少样本学习和跨领域迁移任务中表现出色，成为提示词工程的重要发展方向。

## 2. from CoT Reasoning to Language Agents <https://arxiv.org/abs/2311.11797>

### 2.1. CoT 范式

**思维链**（Chain of Thought, CoT）是一种用于增强大型语言模型推理能力的技术，通过逐步分解问题，并在回答中包含中间的推理步骤，以此帮助模型更好地理解 and 解决复杂问题。显式地输出中间推理步骤不仅可以增强大模型的逻辑、常识和推理的质量，也可让使用者了解模型的思考过程，提高了大模型推理的**可解释性**。

应用 CoT 技术，LLM 可以根据指令 (Instruction) 来理解问题描述和输出格式，并逐步输出逻辑依据 (Rationale)，包括问题的解决方案、中间推理步骤以及与问题相关的任何外部知识等，最后得到答案 (Answer)。模型通过构建少样本示例 (Exemplars)，用来提示输入输出对的基本格式 (不是监督学习)，或者使用 Zero-shot 的方法，通过 "Let's think step by step" 来唤醒大模型的推理能力。



CoT 的实现范式

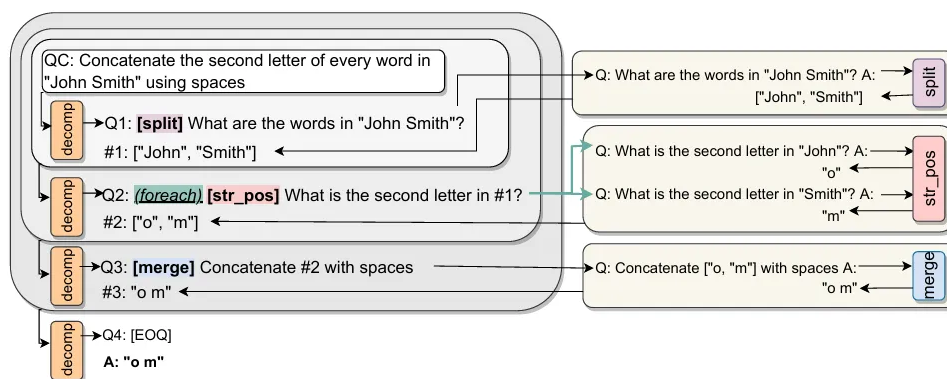
在提示工程框架中，除了传统的链状结构 CoT 之外，也可通过思维树 ToT 或思维图 GoT 来进行逐步推理。后两者通过潜在推理步骤的候选列表和可能性评估来更有效率地搜索推理路线，适合更加复杂的知识推理任务。除此之外，还包括思维框架 (Skeleton-of-Thought) 和程序思维 (Program-of-Thoughts) 等 LLM 推理方法，通过不同的侧重来适应更多下游任务，例如添加回溯和减少端到端生成延迟等，为提示词的设计提供了更多应用框架。

## 2.2. 子任务分解 – Question Decomposition (<https://arxiv.org/abs/2210.02406>)

子任务分解 (Question Decomposition) 是指将一个复杂的查询或问题拆分成更小、更易于处理的子问题的过程，以此来提高理解和解决复杂问题的能力。分解任务有助于 LLM 更好地理解问题背景及其内在逻辑关系，特别是在处理需要多步推理的问题时，通过逐步解决每个子问题，可以减少错误累积的风险，提高最终答案的准确性。

Decomposition Prompt 的核心思路是在 prompt 中把问题进行拆分为子问题，让模型去从上下文中学。与传统的少量提示一样，子任务分解的目标是教会一个 LLM 使用一组小的上下文示例

$D = \{E_1, \dots, E_{|D|}\}$  来找到一个查询  $Q$  的答案  $A$ ，答案  $A$  是从底层分布  $p(A|Q, D, \theta)$  中获得的。在 CoT 风格的 prompt 中，首先需要生成一系列中间推理步骤  $T$ ，然后根据  $T$  得出最终答案，为此需要复杂的上下文示例  $E_j = (Q_j, (T_{j,1}, \dots, T_{j,k}), A_j)$ 。



DECOMP 的推理过程

而在 DECOMP 中，核心是一个分解器 LLM，它试图通过为复杂任务生成一个提示程序 P 来解决该任务。P 的每个步骤都将一个更简单的子查询指向系统可用的辅助子任务函数集 F 中的一个函数。**分解器** (Decomposer) 的少样本提示仅描述了解决复杂任务的子任务序列，将分解后的各个任务委托给特定的子任务 LLMs 进行处理。这些**子任务处理程序**只需较为简单的任务示例集，因此比单独构建复杂任务的示例更加方便，且可以跨多个任务共享，模块化的设计方便改进和交换而不影响整体框架。

## 2.3. 反思-React框架 – Verification and Refinement

在提示词工程中，**反思** (Reflection) 是指促使 LLM 观察其过去的步骤 (以及来自工具/环境的潜在观察)，以评估所选操作的质量的过程，然后将其用于下游的重新规划、搜索或评估等操作。“**反思-React框架**”的核心在于创建一个闭环系统，在这个系统中，通过不断地检验输出的质量，并据此调整输入 (即提示词的设计)，从而实现语言模型表现的持续优化。

在与反思相关的智能体框架中，**SELF-REFINE** 使用单个 LLM 对输出进行反馈 (Feedback) 和改善 (Refine)，不断迭代直到满足停止条件 (指定的时间步或反馈得分)；**Reflexion** 框架包括三个模型，Actor 生成文本和动作，Evaluator 对输出打分，然后通过 Self-Reflection 来生成文字增强反馈来帮助 Actor 进行自我提升；**CRITIC** 通过 text-to-text API 与外部工具交互来核实输出并反馈给 LLM 进行改正。没想到这么简单的切入点也可以水出来三篇论文，给人一种我上我也行的感觉。

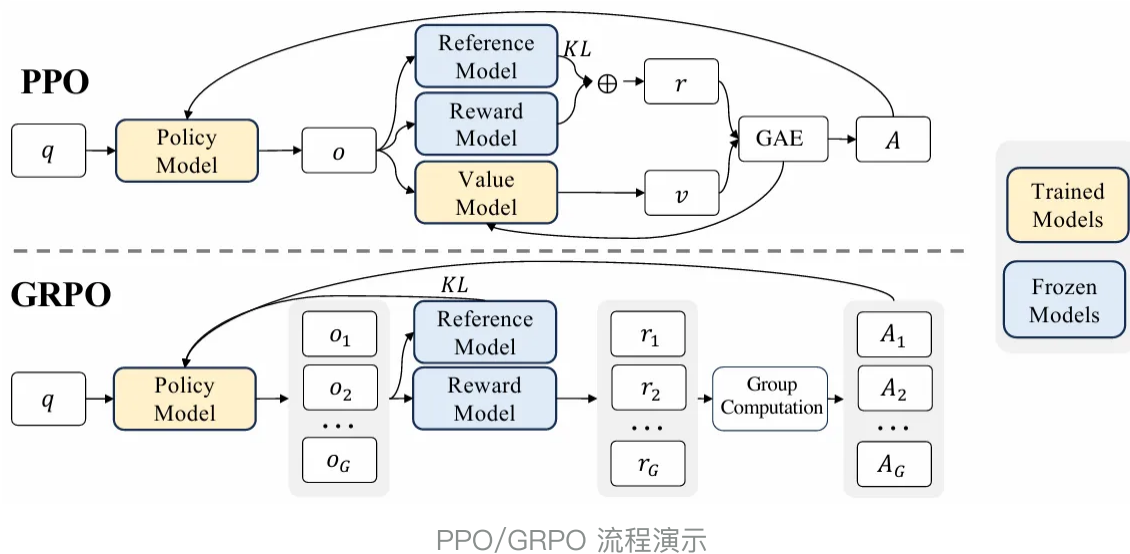
## 2.4. Proximal Policy Optimization & GRPO – 动态调整交互策略

**近端策略优化** (Proximal Policy Optimization, PPO) 是一种基于策略梯度的强化学习算法，通过限制策略更新的幅度，例如裁剪概率比或约束 KL 散度，来确保新旧策略之间的差异不会过大，提升训练的稳定性和收敛性 (<https://arxiv.org/abs/1707.06347>)。PPO 限制了新旧策略的概率比在  $[1 - \epsilon, 1 + \epsilon]$  范围内 (通常  $\epsilon = 0.2$ ) 来避免策略更新过激，因此是一种 "Proximal" 的方法。由于其易于实现且在许多任务上可以取得较好的效果，目前仍是强化学习和知识工程中经常采用的主流算法之一。

应用 Reflection 的 agentic 模式可以对模型过往的输出结果进行评估和对比，进而微调自身或者回溯结果。这种思路有些类似**强化学习**，而在 Deepseek 中正是利用了类似的**群体相对策略优化** (Group Relative

Policy Optimization, GRPO) 算法来提升复杂任务推理性能和降低训练成本的

(<https://arxiv.org/abs/2402.03300>)。GRPO 通过组内相对优势估计和动态正则化直接替代价值函数，对于每个输入问题，模型会基于当前策略生成多个输出，并使用奖励模型进行评分，以平均奖励作为基线，根据相对优势来更新模型策略。



### 3. 参数高效微调 (PEFT) <https://arxiv.org/abs/2403.14608>

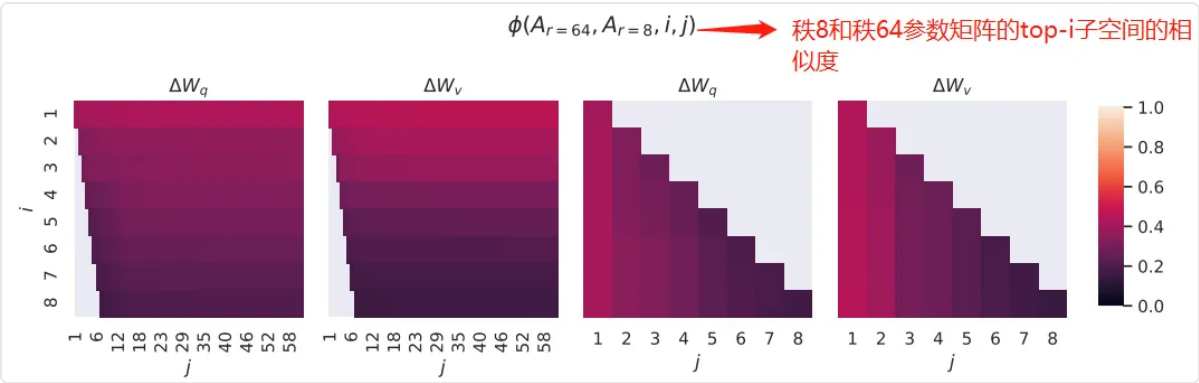
#### 3.1. PEFT Methods for PLMs

**参数高效微调** (Parameter-Efficient Fine-Tuning, PEFT) 旨在通过优化大规模预训练语言模型 (PLMs) 的微调过程，显著降低计算和存储开销，同时保持模型性能。传统微调方法（如全量参数更新）需调整所有模型参数，导致高昂的资源消耗，尤其在参数规模达数十亿级时。PEFT通过仅调整少量参数或引入轻量级模块，实现对模型的高效定制化适配。例如，Adapter方法在Transformer层中插入小型适配器模块（由两层全连接网络构成），仅更新适配器参数；Prompt Tuning则通过设计前缀或后缀提示向量，引导模型生成目标输出。这些方法的核心目标是在不改变原始模型权重的前提下，通过参数冻结或增量学习策略，降低微调成本，同时保留模型的泛化能力。

PEFT方法的多样性源于其对不同任务和硬件约束的适应性。例如，基于**软提示**的PEFT（如 Prefix Tuning）通过优化可训练的前缀嵌入向量，隐式调整模型行为；而基于**稀疏注意力机制**的方法（如 Sparse Tuning）则通过动态选择关键注意力头进行更新。此外，PEFT还支持分布式微调场景，例如仅更新特定层的参数（如最后一层分类头），或结合知识蒸馏技术将大模型的知识迁移到小型学生模型。研究表明，PEFT在自然语言处理 (NLP)、计算机视觉 (CV) 等下游任务中，能够以较低参数更新量（通常<1%）达到接近全量微调的性能，显著提升资源受限环境下的实用性。

#### 3.2. 低秩适配 (LoRA) <https://arxiv.org/abs/2106.09685>

**低秩适配**（Low-Rank Adaptation, LoRA）是一种基于矩阵分解的PEFT技术，通过引入低秩矩阵增量来近似原始权重的更新。具体而言，LoRA假设预训练模型的权重矩阵  $W \in \mathbb{R}^{d \times k}$  的微调变化可以表示为两个低秩矩阵  $A \in \mathbb{R}^{d \times r}$  和  $B \in \mathbb{R}^{r \times k}$  的乘积（ $r \ll d, k$ ），即  $\Delta W = AB$ 。这种低秩约束大幅减少了需训练的参数数量（从  $d \times k$  降至  $r(d + k)$ ），同时通过矩阵乘法保持模型的前向计算效率。LoRA的数学基础源于矩阵近似理论，其有效性依赖于预训练模型权重的冗余性——低秩分解能够捕捉主要变化方向，而忽略次要扰动。



更新参数矩阵存在极小的内在秩，在实际操作中应当将可微调参数分配到多种类型权重矩阵中，而不应该用更大的秩单独微调某种类型的权重矩阵

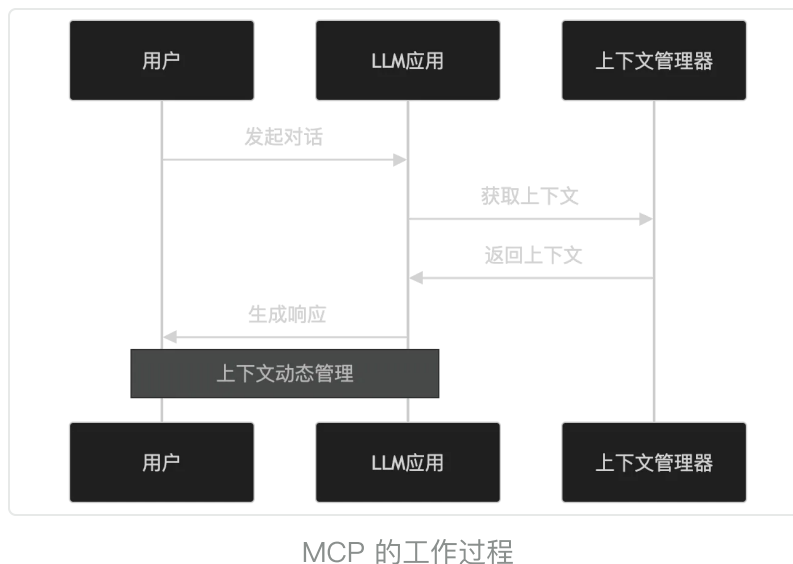
LoRA 的实现通常通过冻结原始权重矩阵  $W$ ，仅优化低秩矩阵  $A$  和  $B$ 。其优势在于：1) 无需修改模型结构，兼容性强；2) 训练时内存占用显著降低，适合边缘设备部署；3) 理论上，低秩假设可抑制过拟合风险，提升模型鲁棒性。实验表明，LoRA在文本分类、机器翻译等任务中，以  $r = 64$  的秩值即可达到全量微调的90%以上性能。此外，LoRA支持多任务适配，通过共享低秩矩阵或引入任务特定的适配模块，进一步扩展其适用性。近期研究还提出动态LoRA（如LoRA-D）和混合精度训练策略，以平衡精度与效率，推动其在工业场景中的落地。

4. 模型上下文协议 (MCP) <https://arxiv.org/abs/2503.23278>

**模型上下文协议**（Model Context Protocol, MCP）是一种旨在优化大规模语言模型（LLM）对上下文信息处理效率的框架，其核心目标是通过结构化协议设计，提升模型对长序列输入的适应性、推理效率及跨任务泛化能力。MCP 的核心是 **模型上下文**，即 LLM 在运行过程中所需的所有外部信息和工具，通过定义标准化的接口和协议，使 LLM 能够动态访问和集成以下内容：

- 1. **外部数据源**：如数据库、API、文档库等，为 LLM 提供实时或历史数据。
- 2. **工具和服务**：如计算工具、搜索引擎、第三方服务等，扩展 LLM 的功能。
- 3. **上下文管理**：动态维护 LLM 的对话上下文，确保连贯性和一致性。





## 4.1. MCP Workflow

某种程度上来说，MCP 类似于计算机网络中的通讯协议，也是通过定义标准化的数据格式来实现 LLM 与外部资源的交互的。MCP 使用 JSON-RPC 2.0 作为消息格式，通过标准的请求、响应和通知消息进行通信；支持多种传输机制，包括本地的标准输入/输出（Stdio）和基于 HTTP 的服务器发送事件（SSE）；MCP 的生命周期包括初始化、运行和关闭三个阶段，确保连接的建立、通信和终止都符合协议规范。

在 MCP 的工作流中，LLM 应用首先会根据任务需求，向外部资源发送请求，外部资源返回所需的数据或服务结果，LLM 应用再将外部资源返回的上下文数据集成到模型中，用于生成响应或执行任务。基础协议定义了三种基本消息类型，分别是请求（Requests）、响应（Responses）和通知（Notifications），通过定义统一的接口和协议，确保 LLM 与外部资源的兼容性，同时显著扩展 LLM 的功能。

## 四、实验过程

### 1. 实验环境

操作系统：Windows 11 + CUDA

依赖工具：

- Docker (用于部署 Dify 和 SearXNG)
- Ollama (用于运行 DeepSeek、Qwen 等大模型)
- Python (用于 SearXNG 自定义脚本或 API 调用)

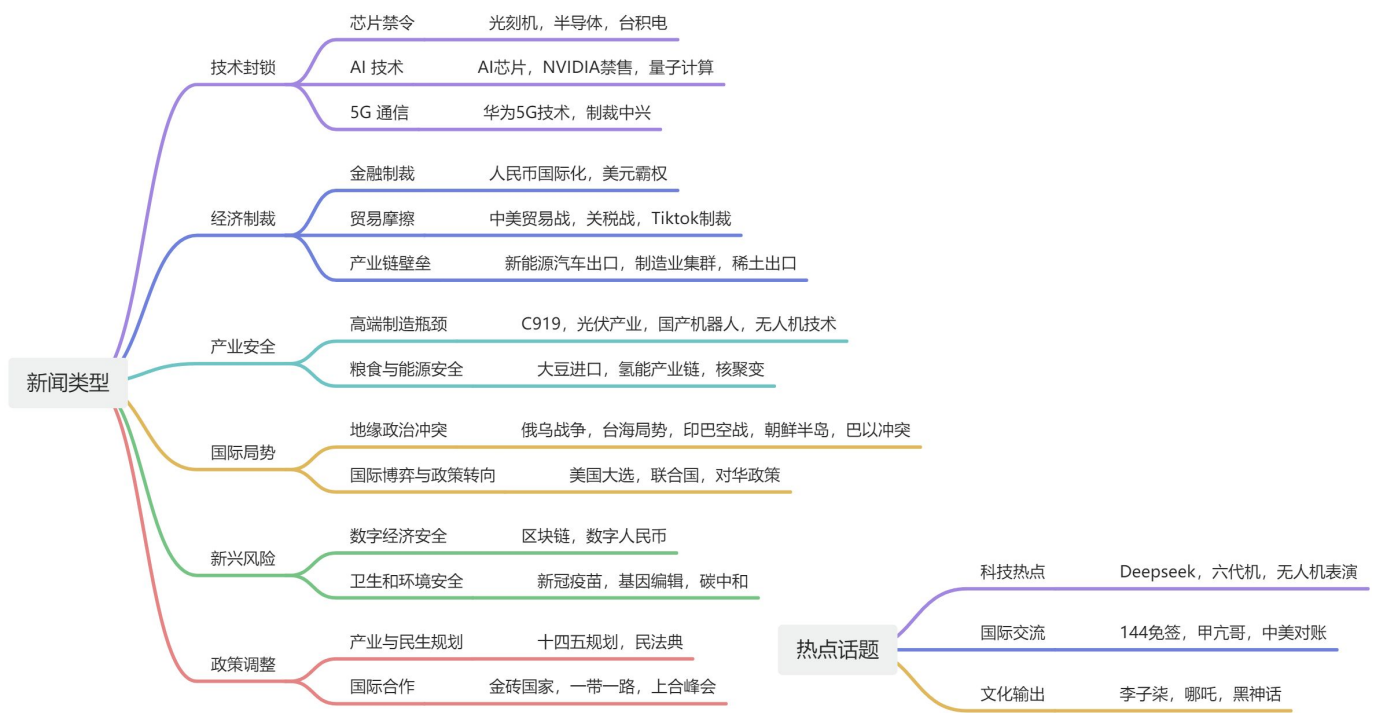
### 2. 数据采集与交叉验证

新闻数据的来源包括传统媒体、社交论坛以及在线新闻平台等。为了综合各种渠道获取的新闻信息，同时保证数据的质量，我们需要根据新闻来源的权威性进行筛选，并对引用进行溯源。考虑到任务要求能够尽量提升数据的丰富程度，最好先通过 LLM 根据相关要求生成一些可行的查询关键词，再通过 **SearXNG** 和 **智谱 WebSearch API** 等工具，以实验内容要求的关键词进行批量抓取，获取原始文本、发布时间及来源链接。然后对数据进行初步筛选：基于哈希值或文本相似度算法剔除重复内容、过滤广告评论等非新闻类文本，并验证发布时间的合理性，以保证新闻的时效性。

searxng 的搜索展示，搭建过程在上个实验中已经详细展示过了

为了确保数据来源可靠，我们优先采取权威性较高的内容来源，例如政府官网、学术期刊及主流媒体等。下一步可采用多维度交叉验证策略：**第一**，通过溯源技术确认新闻事件的关联性，找到新闻引用链的源头，排除 AI 生成或篡改的内容；**第二**，对同一事件的多源报道进行一致性比对，排除网络谣言的影响；**第三**，引入时效性筛选机制，通过历史数据回溯验证长期事件的连贯性，选择最早发布的新闻来源。由于目前人工智能和谣言检测技术还无法较为满意地完成这类目标，且实验要求的 50 篇新闻数据较少，所以这一步还是采用了人工筛选的方法以提高准确度。最终将清洗后的数据转换成 JSON 格式存储，标注来源、时间、关键词等元数据，以搭建可用于后续分析的高质量数据集。

按照常见新闻类型和近几年的热点话题，可以选取以下关键词来抓取新闻，以保证数据集的全面性，额外添加了九个热点话题以增加数据集的时效性：



### 3. 提示词与 Few Shot 示例

本实验需要考察的抽取结果包括以下四个难度递增的维度，通过设计多个提示词的方法分批进行生成：



类型	抽取要素	内容描述	输出格式
元数据	时间	事件发生或提及的具体时间（精确到年/月/日）	JSON 元素
	地点	事件发生或涉及的地理区域（城市、省份、国家）	
	行业	事件所属的行业领域（如科技、金融、医疗）	
	来源	国际媒体/新华网/央视网/人民网/地方媒体	
	体裁	消息/通讯/特写/专访/评论/专题报告/回忆录	
	关键词	抓取新闻事件中的关键词: 这是一篇关于什么的新闻	
信息摘要	主体、客体及其行为	事件发起或主导的对象和事件影响或作用的对象	SPO 三元组
	权威信息	复述涉及政策/法案/条款的相关内容	text: "来源: 内容"
	数据支撑	事件中引用的统计数据、研究报告或权威机构结论	
事件分析	事件背景	事件发生的历史背景、相关社会/政治/经济环境	两三段概述文本
	影响范围	选择社区级/城市级/省级/国家级/世界级	影响范围: xx级
	利益相关方	事件中直接受影响的群体或机构 (如企业、政府、公众、国际组织)	受益: 主体/受害: 主体
内容推理	情感倾向	输出 -5 (极负) 到 +5 (极正) 的情感打分	情感倾向: 数值
	未来影响	预测可能导致的未来影响 (保证客观性)	两三段概述文本
	媒体表现	新闻事件在多来源的不同媒体中的报道倾向、框架或选择性呈现	来源: 一段话概述

- 根据以上表格内容设计的提示词内容，为了让生成内容更加准确，**第一**，需要通过系统提示词让 AI Agent 扮演合适的角色以适应该任务，格式如下所示：

▼ System Prompt JSON |

```
1 {
2   "system_prompt": "你是一个**新闻事件智能分析师**，职责是解析专业领域的内容，严格遵循用户提供的格式要求返回结果，需要保持绝对客观性，所有结论必须直接来自原文，不添加主观判断或外部知识，对矛盾信息选择最相关结果，不进行推测"
3 }
```

- **第二**，在用户提示词中需要明确限制条件，例如存在多个相关时间或主体客体时该如何选择，如果无法提取某个字段应该输出空字符串而不是靠幻觉猜测，此外还要提示禁止引入外部信息，不要在返回内容中加入自己思考或解释的内容等等；

#### ▼ User Prompt

JSON

```
1 {
2     "role": "user",
3     "content": [
4         {
5             "type": "text",
6             "text": "根据提供的新闻文本完成下述任务：\n 1. **三元组提取**:\n - 精准识别事件的主导主体(发起者/决策者)和受影响的客体(对象/领域)\n - 提取主体与客体之间的核心关系(动词短语)，要求关系描述不超过10个字\n - 若存在多个主体/客体，选择相关度最高的单一实体\n - 三元组格式：subject,action,object (逗号分隔，无空格)\n 2. **政策条款提取**:\n - 提取文本中明确提及的政策名称、法案条款或政府声明\n - 格式要求：来源：内容\n - 来源需包含机构名称/文件名，若无明确政策信息，返回空字符串\n 3. **数据证据提取**:\n - 提取文本中引用的统计数据、研究报告或权威机构结论\n - 格式要求：来源：内容\n - 来源需包含机构名称/报告年份，若无明确数据支撑，返回空字符串\n **限制条件**:\n - 严格遵循原文信息，禁止引入外部知识\n - 若无法提取某字段，返回空字符串\"\"而非猜测\n - 不要在返回内容中加入思考或解释的内容\n - JSON输出必须严格符合下述格式要求",
7             "format": "responseFormat_2.json",
8             "sample": "outputSample_2.json",
9             "news": "news.txt"
10        }
11    ]
12 }
```

- **第三**，设计 Few Shot 步骤，本实验通过 responseFormat\_X.json 来规定模型的输出格式，并且提供了 outputSample\_X.json 来展示输出示例，以上文提到的任务分类为例，下面展示了完成本任务所用的相关 json 输入：

#### ▼ responseFormat\_1.json

JSON

```
1 {
2     "date": "YYYY-MM-DD格式的日期字符串（如'2025-07-05'）",
3     "location": "地理区域列表（如['北京市','中国']）",
4     "industry": "行业分类列表（如['科技','金融']）",
5     "source": "来源字符串（如'新华网'）",
6     "genre": "体裁字符串（如'消息'）",
7     "keywords": "关键词列表（如['芯片禁令','ASML','光刻机']）"
8 }
```

## ▼ responseFormat\_2.json

JSON |

```
1 {
2   "subject": "一个词语概括新闻主体（如'欧盟委员会'）",
3   "object": "一个词语概括新闻客体（如'AI系统'）",
4   "action": "不超过10字的关系描述（如'实施监管'）",
5   "etc.": "若存在多个主体/客体，选取相关度最高的一个",
6   "entity_triplet": "格式：subject,action,object（如'欧盟委员会,实施监管,AI系统'）",
7   "authoritative_info": "格式：来源：内容（如'欧盟委员会：AI法案第7条'）",
8   "data_evidence": "格式：来源：内容（如'世界银行2024：GDP增长3.2%'）"
9 }
```

## ▼ responseFormat\_3.json

JSON |

```
1 {
2   "event_context": "事件背景描述（2-3段文字概括,不超过500字）",
3   "impact_scope": "影响范围：xx级（从'社区','城市','省','国家','世界'中选择）",
4   "stakeholders": [
5     "受益：主体",
6     "受害：主体"
7   ],
8   "etc.": "若涉及多个主体,以列表形式列出,如['受益：主体1','受益：主体2']"
9 }
```

## ▼ responseFormat\_4.json

JSON |

```
1 {
2   "sentiment_score": "情感倾向：-5到+5的整数（如'情感倾向：+3'）",
3   "future_impact": "2-3段文字概述未来影响（如'政策实施可能导致产业链重组...'）",
4   "media_representation": [
5     "来源：一段话概述",
6     "来源：一段话概述"
7   ]
8 }
```

考虑到四个子任务的难度不同，在评估结果的时候可以分别进行测试，将输出内容拼贴后就能得到抽取的新闻要素：

```
1  {
2      "date": "2025-07-05",
3      "location": [
4          "北京市",
5          "中国"
6      ],
7      "industry": [
8          "科技",
9          "金融"
10     ],
11     "source": "新华网",
12     "genre": "消息",
13     "keywords": [
14         "欧盟委员会",
15         "ASML",
16         "光刻机"
17     ],
18     "subject": "欧盟委员会",
19     "object": "AI系统",
20     "action": "强制审查",
21     "entity_triplet": "欧盟委员会,强制审查,AI系统",
22     "authoritative_info": "欧盟委员会：人工智能法案草案，欧洲议会：数字服务法案（2023）",
23     "data_evidence": "世界银行：全球AI市场规模达1200亿美元"
24 }
25 }
```

#### 4. 角度 1：基于 Qwen3 和 Deepseek 推理模式设计用户提示词

对本实验给出的四种角度进行分析，不同的实现方法各自有其优点和局限性所在：

1. **网页端大模型**：操作最方便，但是受限于上下文长度，一次只能处理两三篇新闻
2. **api 调用大模型**：可以通过提示词设计实现批量处理，适合一定规模的流水线任务
3. **本地部署大模型**：无 api 调用开销，可以离线运行，但受限于本地环境的算力
4. **微调开源大模型**：更适合下游任务，但训练开销较大，且需要高质量的标注数据

此外，前两种方法调用的大模型默认采取联网搜索的方法优化输出内容，同时也带来了一定的幻觉问题，对本实验的任务来说并不是特别合适；采用本地部署方法可以自定义模型的搜索范围，例如搭配 `searxng` 进行联网搜索或者配置本地知识库进行搜索，因此更加可控。

接下来，我需要确定各个字段的内容。日期应该是2024-10-21，因为活动是在10月21日举行的。地点包括北京市（人民网的总部可能在北京，但活动在重庆渝北，所以可能需要同时包含重庆和中国？但之前的例子中用户将“北京市”和“中国”都放在location里，所以这里可能需要重庆渝北和中国）。行业方面，汽车制造和通信科技是主要的。来源是人民网。体裁是消息。关键词需要包括5G、未来制造、数智工厂、联通、长安汽车、华为、C2M、工业互联网等。

需要注意用户之前的例子中，location有时包含两个元素，比如“北京市”和“中国”。这里活动在重庆渝北举行，所以应该包括“重庆市渝北区”和“中国”。不过用户之前的例子中没有具体的区，可能只需要市级，所以可能写“重庆市”和“中国”。另外，行业方面，除了“科技”和“制造业”之外，这里可能更具体的是“汽车制造”和“通信科技”。但用户之前类似的新闻用了“科技”和“制造业”，所以可能保持一致。

Qwen3 的思考过程，在输入中提示它如果忘记了要求，可以参考前几次输出的内容进行回答

可以体会得到，Qwen3、Deepseek 等网页端大模型足够“聪明”，以至于可以通过 few shot 方法，利用前几次回答的结果便能自己推断出任务要求是什么。在实际处理的时候只需要提示仿照前面的输出进行回答即可，不需要每隔几个轮次重复一遍任务要求，因此并没有遇到前文提及的局限性。这点非常神奇：虽然模型早就忘了一开始提出的任务要求，但是仍然可以依据回答的“惯性”一直保持较为正确的输出，某种程度上来说这也是上下文长度有限的一种解决方案。

## 5. 角度 2：通过 api 调用闭源大模型

将 50 篇新闻转存为一个文件方便读取数据集，根据官方文档查找大模型的 api 端点，并根据文档格式设计请求体，一个包含提示词和配置参数的 JSON 对象：

```
1  from openai import OpenAI
2  import os
3
4  # 初始化OpenAI客户端
5  client = OpenAI(
6      api_key=os.getenv("DASHSCOPE_API_KEY"),
7      base_url="https://dashscope.aliyuncs.com/compatible-mode/v1",
8  )
9
10 messages = [{"role": "user", "content": ""}]
11
12 completion = client.chat.completions.create(
13     model="",
14     messages=messages,
15     extra_body={"enable_thinking": True}, # 开启思考过程
16     stream=True,
17     # stream_options={
18     #     "include_usage": True
19     # },
20 )
```

```

1 reasoning_content = "" # 完整思考过程
2 answer_content = "" # 完整回复
3 is_answering = False # 是否进入回复阶段
4 # print("\n" + "=" * 20 + "思考过程" + "=" * 20 + "\n")
5
6 for chunk in completion:
7     if not chunk.choices:
8         print("\nUsage:")
9         print(chunk.usage)
10        continue
11
12    delta = chunk.choices[0].delta
13
14    if hasattr(delta, "reasoning_content") and delta.reasoning_content is not None:
15        if not is_answering:
16            print(delta.reasoning_content, end="", flush=True)
17            reasoning_content += delta.reasoning_content
18
19    if hasattr(delta, "content") and delta.content:
20        if not is_answering:
21            print("\n" + "=" * 20 + "完整回复" + "=" * 20 + "\n")
22            is_answering = True
23            print(delta.content, end="", flush=True)
24            answer_content += delta.content

```

## 6. 角度 3：基于 Dify Workflow 实现本地部署和设计系统提示词

参考实验四的任务内容，部署 ollama 后根据硬件配置选择模型版本并运行 ollama run，第一次运行模型文件会自动下载到本地，本实验共部署测试了四个不同规模的模型进行测试（deepseek-r1:1.5b, deepseek-r1:7b, Qwen3:1.7b, Qwen3:8b）。下载完模型后尝试输入问题，确认模型可用。

```

Windows PowerShell
PS D:\ollama> ollama run qwen3:8b
pulling manifest
pulling a3de86cd1c13: 100% 5.2 GB
pulling ae370d884f10: 100% 1.7 KB
pulling d18a5cc71b84: 100% 11 KB
pulling cff3f395ef37: 100% 120 B
pulling 05a61d37b084: 100% 487 B
verifying sha256 digest
writing manifest
success
>>> 简单介绍一下你自己
Thinking...
好的，用户让我简单介绍一下自己。首先，我需要确定他们想要什么样的介绍。可能他们只是好奇，或者想了解我的功能和用途。我需要保持回答简洁明了，同时涵盖关键点。

首先，我的名字是通义千问，属于通义实验室，这是阿里巴巴集团的一部分。然后要提到我的主要功能，比如回答问题、创作文字、逻辑推理、编程等。还要说明我的训练数据和语言支持，比如中文和英文，以及覆盖的领域，比如科技、文化、生活等。另外，用户可能想知道我的应用场景，比如日常助手、学习、工作等。最后，要邀请用户提问，保持友好和开放的态度。

需要确保信息准确，没有技术术语，让不同背景的用户都能理解。同时，保持语气自然，避免过于正式。可能用户没有说出来的需求是希望了解我是否能帮助他们解决具体问题，所以结尾部分可以强调我的帮助能力，鼓励他们提问。
...done thinking.

你好！我是通义千问，是通义实验室开发的超大规模语言模型，属于阿里巴巴集团旗下的通义实验室。我的目标是通过理解和生成自然语言，帮助用户完成各种任务，比如回答问题、创作文字、逻辑推理、编程、数据分析等。我支持多种语言，包括中文和英文，并且能够覆盖科技、文化、生活等多个领域。无论是日常交流、学习工作，还是创意灵感，我都可以成为你的得力助手。有什么问题或需要帮助的地方吗？😊

>>>
PS D:\ollama>

```

CLI 展示 ollama 部署大模型



克隆 Dify 代码仓库，根据 Github 提供的官方参考教程，修改 .env 配置文件，启用自定义模型并配置 Ollama 地址，并添加 Docker 内部网络的主机 IP: host.docker.internal。

```
1 wslconfig /setdefault Ubuntu-24.04
2 wsl -d Ubuntu-24.04
3 git clone https://github.com/langgenius/dify.git
4 cd dify
5 cd docker
6 cp .env.example .env
7 vi .env
8 CUSTOM_MODEL_ENABLED=true
9 OLLAMA_API_BASE_URL=http://host.docker.internal:11434
```

运行 `docker compose up -d`，等待所有容器启动完成后，访问 <http://localhost:80> 设置管理员账号并登录。在插件部分安装 Ollama 和 searxng，添加 DeepSeek 模型。首先进入 设置 > 模型供应商，选择 Ollama，填写模型名称和 API 地址 <http://host.docker.internal:11434>。保存配置后，设置默认推理模型并创建新智能体。如果需要联网搜索，可以登录 Dify 控制台，进入 工具 > SearXNG，填写 SearXNG 的访问地址 <http://host.docker.internal:8081>，点击 授权并保存，确认集成成功。

在 Dify 中可以通过控制台 > 应用 workflows 或模型设置 > 高级参数来调整模型参数和系统 prompt，也可以用 json 格式调用 api 来设置。在 Dify 控制台的 "即时测试窗口" 可以进行验证，观察不同参数对输出的影响：

- **Temperature**：控制输出的随机性和多样性。对推理问题可设为 0.1–0.3，通用 QA 可设为 0.4–0.6，故事创作可设为 0.7–1.0
- **Top-p (Nucleus Sampling)**：限制采样范围，避免低概率词汇。与 temperature 搭配使用
- **Max Tokens**：输出长度一般为模型最大支持的 80%，如果输出报错 Query or prefix prompt is too long，可以适当降低 maxtokens 或压缩提示词
- **Penalty**：Presence Penalty 降低重复 token 的出现概率，Frequency Penalty 按 token 出现次数施加惩罚

在 Dify 的 DSL 配置文件中 (例如 YAML) 可以通过 `system_prompt` 字段定义系统级别的提示词，来作为模型的全局上下文，引导模型生成符合特定角色或任务的内容。

## 7. 角度 4：基于 LoRA 微调开源大模型

LoRA 通过在预训练模型的关键层添加低秩矩阵来实现微调，从而减少参数数量并提高效率。为此我们需要收集一个质量可靠的新闻数据集，并对其进行标注。标注内容应包括新闻的各个要素，例如标题、时间、地点、人物、组织、事件等，由于缺少标注数据用于训练，因此 LoRA 微调很难展开。好在微调所需的标注数据量并没有多少，利用前三个角度得到的标注结果就可以进一步对本地模型进行微调，但是只适合于一些简单的提取任务，例如前面设计的元数据等。并且由于数据量过小，训练效果并没有多么显著 (输出内容大致上与角度 3 持平吧)，因此这里只做方法展示。

选择一个适合的开源 LLM，例如 Hugging Face 上的 bert-base-uncased 或其他大型预训练模型，引入 loralib 库将 LoRA 适配器添加到模型的关键层即可。因此关键是确保修改后的模型可以正常训练，需要提前进行本地测试。此后的训练过程并没有什么特别的内容，从结果可以看出使用了自制数据集的模型是否比预训练模型的效果要更好。

## 五、实验结果

### 1. 实验结果评估

1. 元数据准确度 (横向交叉对比)
2. 信息摘要准确度 (横向交叉对比)
3. 事件分析准确度 (横向交叉对比)
4. 内容推理准确度 (横向交叉对比)
5. 输出格式规范 (由 Qwen-3 官方模型进行评估)
6. 总体参考价值 (由 Qwen-3 官方模型进行评估)

**需要注意的是**，本实验四种角度所采用的模型类别和规模有很大的差距，受算力所限，本地部署和微调只能选用参数量较小的模型，其运行效果客观上肯定弱于通过官方 api 调用的 "满血" 大模型。因此该结果仅是用来展示一些可行的评估方法，并不能代表各种实现方法的表现差距。

### 2. 改进思路

#### 2.1. 长文本或富文本处理

本实验所挑选的新闻内容大多数是较短的官方通讯或消息，因此不会产生上下文不足的问题。对于某些较长的新闻或时评，可以采用分段处理的方式，或者提前交给大模型进行一边总结概括。在如今多媒体时代，新闻报道也更趋向富文本的形式，有很多媒体会选择图文并茂或视频的形式来发布新闻，我想这种多模态的处理会是现在的一个重点话题。

#### 2.2. 内容链接与可视化

在文章中链接提取的事件要素，提高准确度，对提取内容进行可视化展示

## 六、深度分析

### 1. Why think step-by-step? 思维链的应用场景

思维链的核心价值在于通过分步推理提升模型对复杂问题的解决能力，其应用场景涵盖数学问题求解、常识推理、科学分析、编程辅助及多模态任务等。例如在数学领域，CoT 通过逐步展示算术运算或代数推导过程，显著降低计算错误率；在医疗诊断中，模型可结合患者症状和检查数据，分步骤推导可能的疾病结论，增强决策透明性。此外，CoT 在教育场景中被广泛用于智能辅导系统，帮助学生理解解题逻辑，而非仅提供答案。

分步推理的优势不仅体现在准确性上，还增强了模型的可解释性。传统模型直接输出结果可能掩盖中间逻辑，而CoT通过显式推理链，使用户能够验证每一步的合理性。例如，在法律咨询中，CoT可引导模型结合案件事实与法律条文，逐步推导责任归属，从而提升用户对模型结论的信任度。这种结构化推理能力使CoT成为复杂决策场景（如金融风险评估）的关键工具。

尽管CoT在复杂任务中表现优异，但在一些 "依靠直观" 的场景下并不一定是最好的选择，例如简单直接的问题、小规模或资源受限的应用场景等，应用思维链也会拖慢模型推理生成的速度，占用大量上下文长度和 token。在本实验中开启深度思考完成一轮回答需要消耗 100k 左右的 token，对于大规模部署和应用场景来说是一个很大的计算负担，因此还是要根据具体情况量力而行。

## 2. Few-Shot 与范例生成

Few-Shot 学习通过少量输入-输出示例引导模型生成目标输出，是提升模型泛化能力的关键技术。例如，在自然语言处理中，Few-Shot 提示模板（如定义数学符号规则）可帮助模型快速适应新任务。动态 Few-Shot 方法进一步优化效果，通过向量存储选择与当前输入最相关的示例，减少无关信息干扰。这种方法在低资源场景（如罕见语言翻译）中尤为有效，显著降低了数据标注成本。

范例生成的质量直接影响 Few-Shot 性能。研究表明，精心设计的示例需包含典型输入特征与明确输出规则。例如，在代码生成任务中，提供涵盖语法、逻辑和异常处理的示例，可使模型更精准地理解任务需求。然而，Few-Shot 仍面临示例选择偏差和领域迁移限制的挑战，需结合元学习或主动学习策略进一步优化。

在本实验中我发现，流水线式的任务可以将前几次输出的内容作为 fewshot 示例供给后面的任务学习，一定程度上可以解决上下文过长导致遗忘任务要求的问题。但在这一过程中还需要人为进行监督，防止模型由于错误的认知偏差导致输出质量越来越低。例如在前面的任务设计中存在某些空字符串的项，模型会误以为空字符串是 "主人的要求" 导致那一项一直为空，需要额外添加一段简短的提示词进行纠正才可以。

## 3. PEFT 的设计与跨领域应用

参数高效微调通过调整预训练模型中的一小部分参数，实现快速适配新任务，显著降低计算资源消耗。例如 LoRA 通过低秩矩阵投影，仅需 0.19% 的参数即可在 Twitter 投诉分类任务中达到 86.3% 的准确率，接近人类基准。此外，PEFT在视觉领域中同样表现出色，其生成模型的 checkpoint 极大减少了存储与部署成本。

PEFT 的跨领域应用得益于其对预训练模型知识的保留能力，并利用少量，高质量的标注数据来适应下游专家任务。例如，在医疗文本分析中，PEFT 可微调模型以适应专业术语，同时避免灾难性遗忘。然而，其对高质

量预训练模型和标注数据集仍然太过依赖，导致实际应用过程中如何获取数据集反倒成了最大的难题。未来研究需探索更高效的参数选择策略，以平衡微调精度与资源消耗，或者采用元学习等方法彻底摆脱对数据集的依赖。

## 4. MCP 调用方法的优势

Model Context Protocol 通过标准化接口扩展大语言模型（LLM）的功能，使其能够主动调用外部工具。其核心优势在于解耦模型与工具，支持灵活插件化架构。这种设计类似于前端调用后端API，使模型从 "被动输出" 转向 "主动执行"。目前已有许多厂商和模型提供者支持了该协议，通过明确的请求格式、调用ID跟踪和权限控制，MCP可防止模型滥用敏感接口，同时支持多工具协同。这种能力使其成为构建智能体系统（如AI工作流程框架）的核心协议，推动LLM从单点任务工具向通用操作系统演进。

就像其他的网络协议一样，可靠性是靠大家的支持与建设不断积累起来的。也许现在 MCP 在 AIGC 领域的存在感仍然很弱，但在未来 AI Agent 甚至 Agentic System 的场景下，MCP 和它的后继者会成为一个重要的虚拟层，来支持更多大模型工具的开发和应用。比较局限的一点是——这一协议的具体内容对很多开发者来说还是有些晦涩难懂，希望未来 MCP 技术可以让完全不懂 AI 的开发者也能得心应手地完成开发。

# 七、心得体会

## 1. 实现细节

### 1.1. Prompt 调整思路

#### 1.1.1. 参考答案

提示词微调的核心在于通过示例引导和动态反馈优化模型输出模式。在 LLM Prompt 优化方面，可以通过 Few-Shot 方法通过提供少量正确与错误的输出示例，帮助模型识别任务需求与潜在偏差，缺点是会增加 token 消耗，且不一定适用于所有任务。采用反思-React 框架，利用一些 training 示例让模型先学习表达方式，再将输出内容与标签做比较，让模型动态调整输出模式。这种方法的缺点是非常依赖上下文长度，如果没有参数量限制，选择性能更好的模型能大大改善输出结果。

这种思路类似 teacher-student 训练方法。对本实验来说，可以利用效果较好的开放模型 api 来生成一些效果较好的用例，例如基于 ChatGPT、Deepseek、Qwen3 等模型对本实验任务所提供的 35 个问题进行回答，将输出结果用作参考答案。在微调提示词的时候，把本地部署模型输出的内容与参考答案反馈给该模型，让其对两者进行对比评估，并改进输入 prompt。

#### 1.1.2. 信任来源

当本地数据库的知识与联网查找的内容存在冲突时，Agent 应该信任哪一边？这需要建立优先级验证体系以确保结果可靠性。此外，需警惕数据源的隐性偏见。例如，某些开放数据集可能存在标注偏差或过时信息，需

通过统计显著性检验或领域专家审核筛选高质量数据。这种思路类似于人类决策中的“多源求证”，优先采纳时效性高、权威性强的数据源，并通过交叉验证解决矛盾问题，建立层级化信任链来减少错误传播风险。

### 1.1.3. 输出格式限制

如何限制 Agent 的 "长篇大论"，让其记住自己的输出格式？关键在于结构化约束与目标导向提示。明确指令需包含输出形式，并通过惩罚机制抑制冗余生成，进一步需要结合上下文感知设计动态格式规则。例如，针对选择题场景，模型需强制输出“选项A/B/C”而非详细解释；而在开放式问答中，则允许适当扩展但需严格遵循字数限制。这种格式限制本质上是将用户意图转化为可执行的约束条件，从而提升输出的精准度与实用性。在实验过程中我发现，像摘抄概括或提取关键词这类的任务，如果不额外加以限制，模型会变得越来越 "话痨"，因此在输出过程中进行人为干预是必要的，除非每一次生成都重新描述一次提示词限制，这样做会大量浪费 token。

### 1.1.4. 检查和确认

在使用 Qwen3 进行测试时，注意到了思维链中经常会出现 "需要确保"、"再次核对" 等思考内容，通过多源验证与思维链显式化确保输出可靠性。通过引入自动化校验机制，例如在事实核查任务中，可调用外部API验证关键数据，并生成“依据来源X与Y，结论为...”的解释性文本，通过多次对比不同数据源的结果，模型可识别潜在矛盾并调整输出。这种“检查-确认-修正”的闭环流程，是提升模型可信度的关键技术路径。

## 1.2. 大模型的回答"惯性"

在本次实验过程中我收获了一个经验，有时大模型的回答存在某种 "惯性"，可以通过 few shot 方法，利用前几次回答的结果便能自己推断出任务要求是什么。在实际处理的时候只需要提示仿照前面的输出进行回答即可，不需要每隔几个轮次重复一遍任务要求，因此并没有遇到前文提及的局限性。如果大模型可以利用上下文作为参考进行回答，虽然模型早就忘了一开始提出的任务要求，但是仍然可以依据回答的 "惯性" 一直保持较为正确的输出，某种程度上来说这也是上下文长度有限的一种解决方案。

## 1.3. Dify Workflow

在 Dify 中提供了两种任务执行机制：Agent 和 Workflow。**Agent** 是一种具有 "策略+工具调用能力" 的 **AI 执行体**，能够支持多种推理策略自主决策，结合用户输入、历史对话、系统变量做出策略选择，并访问外部 API 或数据库来实现检索；**Workflow** 是一个基于节点可视化的流程编排系统，旨在 "无代码/低代码" 地编排一个复杂的，包含多个步骤与条件判断的 LLM 应用。在 Dify 中可以配置多个 Agent，分别绑定不同功能、策略和触发条件，或者使用 Workflow 来按操作进行编排 (类似 **pipeline** 设计)。两者分别侧重**自主推理**和**流程管控**，针对不同的业务目标，可以选择不同的执行机制或两者联动来实现具体的实现风格。

例如本实验的四个子任务，可以利用实验原理中提到的子任务分解方法，交给四个不同的 AI Agent 来并行执行，最后合并结果并交给另一个模型来进行验证。利用这种集群式的思路可以将复杂的问题转化为几个较为简单的任务，这样 AI Agent 的执行准确率较高，同时也大大节省了处理时间。我想对于很多流水线式的应用任务

来说，利用这种 workflow 技术可以快捷地进行批量处理，非常有应用价值。对于 dify，我想我还要进一步地去深入学习一下，争取能在以后写进自己的简历里)

## 2. 思考与改进

通过本实验我亲身体验了大模型联网智能体的本地部署方式，以及 Dify、Ollama、SearXNG 等工具的使用。相比直接调用 api 的方法，通过 LLM 应用开发平台来部署智能体可以实现更加定制化、流程化的 AI 应用，同时也可提升对模型运行机制和交互逻辑的理解。

在构建多跳推理任务的过程中，我研究了 Agentic System 的实现范式和提示工程技巧，也认识到了合理设计 Prompt 和工具链对于提升智能体推理能力的重要性，对提示工程有了更加深入的认识，这一领域的研究比自己之前想的更加复杂 (虽然提示工程师感觉还是很扯)。现在的 LLM 能力远不及 Agentic 框架所构想的那般 "自主思考解决任务"，但是在尝试了构建 prompt 比较之后，我能体会到无论是系统 prompt 还是自定义 prompt 都对智能体解决问题有着很大的帮助，可能 AI 做事情也需要指导老师... 假设检验分析方法是我在人机交互课程上学到的，现在在智能体设计优化和提示词微调上也能用得到，也算是一种实验研究经验的积累吧。

在本实验中我接触了 Docker 容器使用和联网部署应用的实现方式，获得了宝贵的开发经验。此外我也额外了解了 Docker Compose 的微服务架构下各组件如何协同工作，如果有机会的话我想尝试一下 k8s 是如何编排容器的。容器化技术在应用部署与环境管理方面十分高效与便捷，先前我尝试部署过很多开源模型，每次都会在配置实验环境上耗费特别多的精力。相比来说 docker 算是规范化了环境配置和迁移操作，取代了繁杂的安装说明，对开发者和使用者来说都是节省寿命的好工具，然而在国内迟迟没有在业务项目开发上广泛使用，而且长期在大学课程里缺席，我想这是未来需要改变的一点。

在深度分析环节我对 RAG 方法和知识图谱应用有了更加深入地理解，结合自己之前的学习，我也尝试设计了一些节点聚类的创新方法，进一步深化了先前的研究结果 (虽然只是写了篇综述)。纸上得来终觉浅，实际尝试过部署模型后我能够感觉到自己之前学习到的东西都很浅，无法结合具体实际来解决问题，实践场景中的侧重点像是一门全新的课程。LLM 能够涌现出智能本身就很神奇，谁都不知道一个根据概率预测下一个词的模型是如何真正理解自然语言并给出自己的思考的，也没人能说明构建 prompt 的方式为什么会那么有效，一句 "lets think step by step" 就能让智能体智商暴涨。现在我们造出了 Agent，但是想要真正理解它还离得很远吧...