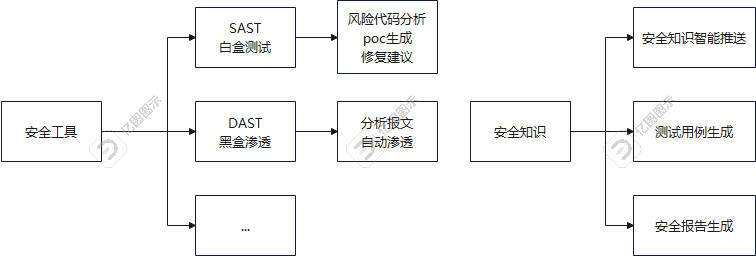
一、ai安全应用场景



二、大模型的核心痛点

1.可控性

模型输出可能偏离用户意图，尤其在复杂任务（多步骤推理、格式约束）中可能出错。

2.幻觉（Hallucination）

大模型可能生成看似合理但事实上错误或虚构的内容（例如捏造数据、事件或引用不存在的文献）。

原因：模型依赖概率生成文本，缺乏对真实性的严格验证。

3.知识更新滞后

大模型的训练数据是静态的（例如GPT-4的知识截止到2023年），无法动态学习新知识或实时事件（如新闻、学术进展等）。

后果：回答过时信息，或对新兴领域（如最新技术）缺乏认知。

4.计算资源与推理成本高

大模型参数量庞大（千亿级），导致推理延迟高、部署成本高昂，难以在资源受限的场景（如移动端）应用。

5.领域泛化能力不足

通用模型在垂直领域（如医疗、法律、金融）的表现较差，缺乏专业术语和逻辑推理能力。

可控性问题原因可能由于用户设计的提示词不够清晰，造成大模型输出结果不符合预期，需要提示词优化

1.提示词工程

1.1提示词的设计准则

1. 清晰性：设定明确的目标

提示词的清晰性是至关重要的，我们需要确保所提出的问题或要求简单、直接而不含歧义。清晰的目标将有助于AI将注意力集中在回答关键点上。

不清晰的目标：如“请谈谈AI对程序员的影响”，AI可能会尝试从多个角度进行分析，导致注意力分散，最终输出的答案笼统，缺乏深度。

清晰案例：而如果我们改为“请分析AI如何提高自动化测试中的软件质量”，AI的注意力会专注于具体领域，获取的答案更为细致。

2. 结构化：提供逻辑框架

一旦确定了目标，结构化的提示词将极大地提高AI的响应效率和逻辑性。结构化的提示可以帮助AI扫描信息，识别和响应问题。

优化提示示例：例如，使用分步骤的结构“请按照以下顺序为我制定一份AI编程产品的需求文档”，随后列出调研、竞品分析、核心需求等步骤，能够明确逻辑框架，提升AI的解题准确性。

3. 细节化：提升答案的相关性与精准度

细节是实现深度分析的关键，清晰地规定输出格式及所需信息的细节，能极大减少AI的信息探索空间。

缺乏细节的提示：如“写一首关于秋天的诗”太过宽泛，AI可能生成的内容过于抽象，缺乏针对性。

细节案例：而如果要求“模仿杜甫的风格以七言绝句写一首关于秋天的诗”，并指定“必须包含‘九九六’与‘需求变更’这两个关键词”，AI的生成会更具针对性、高度相关性。

1.2提示词技术

(1)零样本提示

零样本提示（Zero-Shot Prompting）技术的核心思想是：通过自然语言指令直接引导大语言模型（LLM）完成任务，而无需提供任何具体示例。

输入：

将文本分类为中性、负面或正面。

文本：我认为这次假期还可以。

情感：

输出：

中性

局限性：当任务复杂、歧义性强或需要领域专业知识时（如医学诊断），零样本可能失效，此时需转向少样本（Few-Shot）

(2)少样本提示（Few-Shot Prompting）

少样本提示的核心思想是通过在输入中提供少量示例，引导大型语言模型在上下文学习中理解任务模式并提升复杂任务的表现。

输入：

这太棒了！// 消极

这太糟糕了！// 积极

哇，那部电影太棒了！// 积极

多么可怕的节目！//

输出：

消极

局限性：对多步推理任务（如数学问题）效果有限，模型可能忽视逻辑步骤而依赖表面模式匹配。

当少样本失效时，需转向思维链（CoT）提示（显式分解推理步骤）或模型微调，以解决复杂问题。

(3)链式思考（CoT）提示

链式思考（Chain-of-Thought, CoT）提示的核心思想是引导模型生成逐步推理的中间步骤（即"思维链"），模拟人类的逻辑思考过程，从而提升模型的推理能力。可以将其与少样本提示相结合，以获得更好的结果，以便在回答之前进行推理的更复杂的任务。

输入：

这组数中的奇数加起来是偶数：4、8、9、15、12、2、1。

A：将所有奇数相加（9、15、1）得到25。答案为False。

这组数中的奇数加起来是偶数：17、10、19、4、8、12、24。

A：将所有奇数相加（17、19）得到36。答案为True。

这组数中的奇数加起来是偶数：16、11、14、4、8、13、24。

A：将所有奇数相加（11、13）得到24。答案为True。

这组数中的奇数加起来是偶数：17、9、10、12、13、4、2。

A：将所有奇数相加（17、9、13）得到39。答案为False。

这组数中的奇数加起来是偶数：15、32、5、13、82、7、1。

A：

输出:

将所有奇数相加（15、5、13、7、1）得到41。答案为False。

局限性：推理路径单一，若推理中间步骤出错，最终答案必然错误。

(4)自我一致性

通过少样本 CoT 采样多个不同的推理路径，并使用生成结果选择最一致的答案。这有助于提高 CoT 提示在涉及算术和常识推理的任务中的性能，解决单一推理路径的局限性。

输入：

Q：林中有15棵树。林业工人今天将在林中种树。完成后，将有21棵树。林业工人今天种了多少棵树？

A：我们从15棵树开始。后来我们有21棵树。差异必须是他们种树的数量。因此，他们必须种了21-15 = 6棵树。答案是6。

Q：停车场有3辆汽车，又来了2辆汽车，停车场有多少辆汽车？

A：停车场已经有3辆汽车。又来了2辆。现在有3 + 2 = 5辆汽车。答案是5。

Q：Leah有32块巧克力，她的姐姐有42块。如果他们吃了35块，他们总共还剩多少块？

A：Leah有32块巧克力，Leah的姐姐有42块。这意味着最初有32 + 42 = 74块巧克力。已经吃了35块。因此，他们总共还剩74-35 = 39块巧克力。答案是39。

Q：Jason有20个棒棒糖。他给Denny一些棒棒糖。现在Jason只有12个棒棒糖。Jason给Denny多少棒棒糖？

A：Jason有20个棒棒糖。因为他现在只有12个，所以他必须把剩下的给Denny。他给Denny的棒棒糖数量必须是20-12 = 8个棒棒糖。答案是8。

Q：当我6岁时，我的妹妹是我的一半年龄。现在我70岁了，我的妹妹多大？

A：

输出：

当我6岁时，我的妹妹是我的一半年龄，也就是3岁。现在我70岁了，所以她是70-3 = 67岁。答案是67。

其他提示词案例可参考

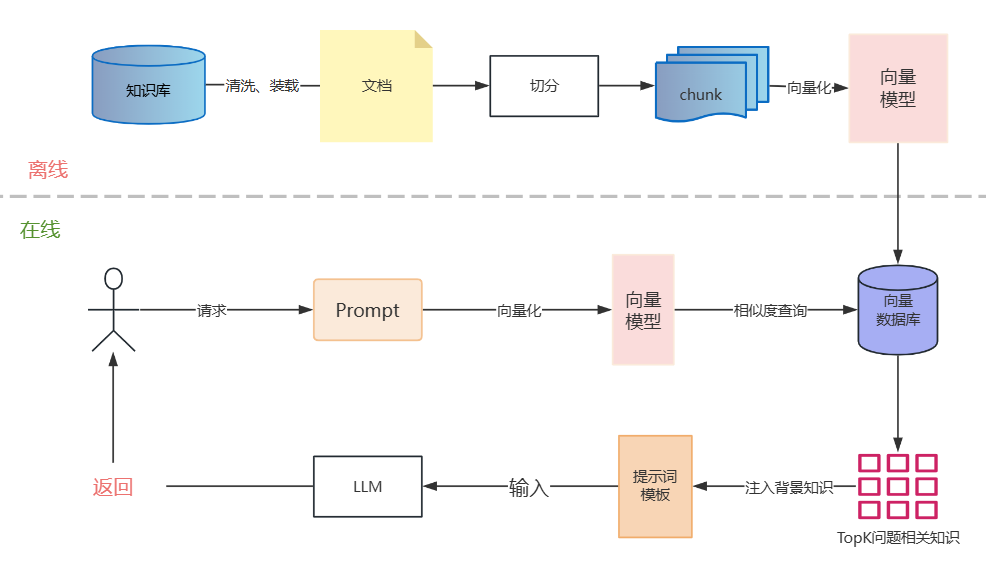
https://www.promptingguide.ai/zh

大语言模型出现幻觉原因是大语言模型训练的数据基于网络上已存在的数据，若提问的问题属于大语言模型的知识盲区，大语言模型不知道自己不知道，就会胡乱回答，通过检索增强生成（RAG）技术，可以解决大模型的幻觉问题。

2.RAG检索增强生成（Retrieval Augmented Generation）

RAG本质就是检索技术与LLM 提示的组合。RAG 会接受输入并在知识库检索出与输入问题相关的文档。这些文档作为上下文和输入的原始提示词组合，送给大语言模型得到最终的输出。RAG 让语言模型不用重新训练就能够获取最新的信息，基于检索生成产生可靠的输出。

简单的rag流程图



rag步骤

1.文档处理阶段

|  |
| --- |
| def process(self, file\_path):  loader = TextLoader(file\_path, encoding='utf-8') # 加载文本文件  text\_splitter = RecursiveCharacterTextSplitter(  chunk\_size=500, # 分块大小  chunk\_overlap=100 # 块间重叠  )  embeddings = OllamaEmbeddings(model=self.model) # 文本向量化  vectorstore = FAISS.from\_documents(chunks, embeddings) # 创建向量数据库 |

(1)文本分块：通过滑动窗口机制（500字符长度+100字符重叠）将长文档切分为语义连贯的小块

(2)向量化：使用Ollama模型生成文本块的向量表示

(3)索引构建：使用FAISS构建高效的向量索引，支持快速相似度检索

2.检索阶段（Retrieval）

|  |
| --- |
| def \_rag\_response(self, question):  retrieved\_docs = self.retriever.invoke(question, {"k": 2}) # 检索最相关的2个文档块  context = "\n".join(set(doc.page\_content.strip() for doc in retrieved\_docs)) |

(1)将用户问题同样转换为向量

(2)在FAISS索引中执行相似度搜索（余弦相似度）

(3)返回top-2最相关文档块作为上下文，文档内容去重处理。

3.生成阶段（Generation）

|  |
| --- |
| def \_call\_ollama(self, question, context):  system\_prompt = """基于上下文回答问题...""" # 系统级指令  user\_prompt = f"""基于以下信息回答问题：{question} 上下文：{context}"""  response = self.client.chat(model=self.model, messages=[...]) |

rag源码



rag优化方法

1.知识库数据数据预处理与清洗

数据质量提升：移除噪声（特殊字符、HTML标签）、纠正拼写/语法错误、去重

2.更好的提词设计

通过为系统给出「如果你不确定答案是什么，就告诉我你不知道」这样的指示，就能鼓励模型承认自己的局限，并更透明地向用户传达它的不确定。

3.对 chunk\_size 和 similarity\_top\_k 进行超参数微调

chunk\_size 和 similarity\_top\_k 这两个参数可用于管理 RAG 模型的数据检索过程的效率和效果。调整这两个参数会影响被检索信息的计算效率和质量之间的权衡。

4.重新排名

先扩大召回再精准排序。初步检索阶段，使用向量相似度算法（例如余弦相似度、点积等），用于从向量数据库中快速检索与问题最接近的文本片段。第二次排序，使用 Cohere 专用的 Rerank 模型（例如 rerank-multilingual-v2.0 等），基于语义相关性对初筛结果重新排序。

5.调整检索策略

依据不同任务场景选择最合适的检索器至关重要

6.对嵌入进行微调

对嵌入层微调，提升语义检索模型的准确度

7.prompt 压缩

上下文过长，非关键信息过多会影响大模型的推断。使用LLMLingua小语言模型来检测和剔除prompt中的不重要token，将其转化为一种人类很难理解但是LLMs能很好理解的形式。

来源：

https://zhuanlan.zhihu.com/p/706873537

3.训练模型

数据集

DeepSeek R1及其蒸馏模型，推理过程的具体体现就是在回复内容中，会同时包含推理部分内容和 最终回复部分内容，并且其推理部分内容会通过（一种在模型训练过程中注入的特殊标记）来进行区分。也就是说， DeepSeek R1模型组的回复格式是一种非常特殊的格式，即包含think部分内容，也包含 response部分内容。

因此，在围绕DeepSeek R1模型组进行微调的时候，微调数据集的回复部分文本也需要是包含推理 和最终回复两部分内容，才能使得DeepSeek R1模型组在保持既定回复风格的同时，强化模型能力，反之则会导致指令消融问题（模型回复不再包含think部分）。

同时包含思考和结果的数据集为COT数据集，类似NuminaMath CoT数据集、APPs（编程数据集）、 TACO（编程数据集）都是COT数据集，均可用于推理模型微调。

数据集结构如下：



一、基础概念介绍

1.1 微调与强化学习、模型蒸馏

伴随着DeepSeek的兴起，关于强化学习训练、模型蒸馏等概念也逐渐被人熟知，这里简单总结下这三者的异同。微调、强化学习训练和模型蒸馏都是常用的技术方法，尽管这些方法在某些方面存在交集，但它们的核心原理和任务目标却有显著差异。

1. 微调（Fine-tuning）：

微调是指在已经训练好的大型预训练模型的基础上，进一步训练该模型以适应特定任务或特定领域的数据。相比从零开始训练一个模型，微调所需的数据和计算资源显著减少；可以在特定任务上取得更好的性能，因为模型在微调过程中会重点学习与任务相关的特性；可以在多种领域（如情感分析、问答系统等）上进行微调，从而快速适应不同应用场景。

举个 ：想象一下，你有一只机器人狗，它已经在基本的狗行为上进行了初步训练，比如行走和听从简单的命令。微调就像是对这只机器狗进行进一步的训练以适应特定的任务环境。比如说，你希望这只机器狗能够在公园里捡回特定种类的球。通过微调，你可以在原有的训练基础上，用一组特定的数据集（比如各种颜色和大小的球）来调整其行为，使其在新环境中表现得更好。

●目标：通过少量的标注数据对预训练模型进行优化，适应具体任务。

●特点：微调的计算量相对较小，能够在有限的数据和计算资源下提升模型在特定任务上的性能。

●应用：常用于下游任务如情感分析、机器翻译、推荐系统等。

微调和rag特点对比

|  |  |  |
| --- | --- | --- |
| 特点对比 | 检索增强生成(RAG) | 微调(FT) |
| 知识更新 | RAG直接更新检索知识库，保持信息最新。模型无需频繁的重新训练，适合动态数据环境。 | FT存储静态数据，知识与数据更新需要重新训练 |
| 外部知识 | RAG擅长利用外部资源，非常适合文档或其他结构化/非结构化数据库。 | 虽然FT可以对大语言模型进行微调以对齐预训练学到的外部知识，但对于频繁更改的数据源来说可能不太实用。 |
| 数据处理 | RAG对数据加工和处理的要求低。 | PT依赖高质量数据集，有限的数据集可能不会产生显著性能提升。 |
| 模型风格 | RAG主要关注信息检索，擅长整合外部知识，但可能无法完全定制模型的行为或写作风格。 | FT允许根据特定的语气或术语调整大语言模型的行为、写作风格或特定领域的知识。 |
| 可解释性 | RAG通常可以追溯到特定数据源的答案，从而提供更高等级的可解释性和可溯源性。 | FT就像黑匣子，并不总是清楚模型为何会做出这样的反应，具有相对较低的可解释性。 |
| 计算资源 | RAG需要高效的检索策略和大型数据库相关技术。另外还需要保持外部数据源集成以及数据更新。 | FT需要准备和整理高质量的训练数据集、定义微调目标以及相应的计算资源。 |
| 延迟和实时要求 | RAG需要进行数据检索，可能会有更高延迟。 | 经过FT的大语言模型无需检索即可响应，延迟较低。 |
| 减少幻觉 | RAG本质上不太容易产生幻觉，因为每个回答都建立在检索到的证据上。 | FT可以通过将模型基于特定领域的训练数据来减少幻觉。但当面对不熟悉的输入时，它仍然可能产生幻觉。 |

综上所述，RAG是外挂一个知识库，模型没有被再训练，适合动态数据、处理幻觉，且可解释性高、通用能力更强

微调，模型被训练，适合给模型定制能力、延迟低

2. 强化学习（Reinforcement Learning）：

强化学习是一种机器学习方法，它通过让智能体在环境中执行动作，以获得反馈或奖励信号，从而学习最优策略。通过不断地试错和调整策略，智能体逐渐找到能够最大化长期回报的行为路径。这种学习方法常用于需要决策和动态环境交互的任务，如游戏、机器人导航和自动化控制系统。

举个 ：强化学习训练则有点像是教这只机器狗通过尝试和错误来学习新技能。在这种情况下，你没有直接告诉它应该怎么做，而是为它设定一个目标，比如尽可能快地找到并捡起一只球。机器狗每完成一次任务都会获得奖励，然后它将通过调整自己的行为来最大化获得的奖励。例如，如果机器狗发现跑直线能更快地找到球，它可能会在未来的尝试中更倾向于这样做。

●目标：通过与环境的交互，学习最优的行为策略，最大化累积奖励。

●特点：强化学习强调动态决策，它通常不依赖于预定义的数据集，而是依赖于与环境的持续交互。

●应用：强化学习在游戏AI（如AlphaGo）、机器人控制、自动驾驶等任务中有广泛应用。

https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.1\_(8B)-GRPO.ipynb

3. 模型蒸馏（Model Distillation）：

模型蒸馏是一种模型压缩技术，通过将一个复杂的大型模型（通常称为“教师模型”）中的知识迁移到一个更小的模型（称为“学生模型”）。在这个过程中，教师模型首先对训练数据进行预测，生成软标签即概率分布。这些软标签包含了有关任务的重要信息。学生模型则使用这些软标签进行训练，以接近教师模型的性能。模型蒸馏能够在保持高精度的同时，显著减少模型的大小和计算消耗，适用于在资源受限的环境下部署机器学习模型。

举个 ：你有一只非常昂贵和精密的机器人狗，它可以完美执行任务。为了降低成本，你希望制造一个更简单的机器狗，同样能有效完成任务。通过模型蒸馏，你会使用大狗的行为数据来训练小狗，让后者理解和模仿前者的精妙动作，同时保持高效性。

●目标：通过教师模型的“知识转移” ，帮助学生模型提升性能，特别是计算能力有限的设备上。

●特点：蒸馏的核心在于知识的迁移，尤其是在模型压缩和部署方面的优势。学生模型通常在性能上能接近教师模型，但参数量更小，计算更高效。

●应用：常见于模型压缩、边缘计算、低功耗设备的部署中，用于提升部署效率并降低计算需求。

1.2 大模型微调

与RAG（Retrieval-Augmented Generation）或Agent技术依靠构建复杂的工作流以优化模型性能不同，微调通过直接调整模型的参数来提升模型的能力。这种方法让模型通过在特定任务的数据上进行再训练，从而'永久'掌握该任务所需的技能。微调不仅可以显著提高模型在特定领域或任务上的表现，还能使其适应于各种具体应用场景的需求。这种能力的增强是通过更精细地调整模型内部的权重和偏差，使其在理解和生成信息时更加精准，因此被广泛用于需要高精度和领域适应性的任务中。

1.2.1 全量微调与高效微调

从广义上讲，微调可以分为两种主要方式：全量微调和高效微调。全量微调是指利用所有可用数据来重新训练模型，以全面优化其参数。尽管这种方法对计算资源的需求较高，但它能够在最大程度上提升模型对特定任务的适应能力。相反，高效微调则采用更精简的策略，只使用部分数据进行调整，并主要修改模型的部分参数。这种方法以相对较低的计算开销，实现对模型性能的显著提升，类似于“以小博大”，非常适合在资源有限的情况下快速调整和增强模型的性能。

全量微调（Full Fine-Tuning）

举个 ：想象一下你在一家公司管理一个团队，这个团队的所有成员已经接受了基础培训，知道如何处理一般的工作任务。现在，公司引入了一个全新的复杂项目，要求团队具备更多的专业技能和知识。

●优点：全面掌握所有相关技能，使模型对新任务有更高的适应性。

●缺点：耗时更长，资源消耗大。

高效微调（Efficient Fine-Tuning）

高效微调的方法更有针对性，它不需要花费大量的时间和资源。举个 ：比如，如果机器人狗的任务只是要学会在一种新环境中识别特别的障碍物，你可以在已有的模型基础上，仅仅微调那些与识别相关的参数，而无需重新训练整个模型。

●优点：节省时间和资源，快速提升特定技能。

●缺点：可能不如全面培训那样细致和彻底，但能够在特定任务中高效达标。

现在绝大多数开源模型，在开源的时候都会公布两个版本的模型，其一是Base模型，该模型只经过了预训练，没有经过指令微调；其二则是Chat模型（或者就是不带尾缀的模型），则是在预训练模型基础上进一步进行全量指令微调之后的对话模型﻿：

1.2.2 高效微调与LoRA、 QLoRA

尽管全量微调可以对模型的能力进行深度改造，但要带入模型全部参数进行训练，需要消耗大量的算力，且有一定的技术门槛。相比之下，在绝大多数场景中，如果我们只想提升模型某个具体领域的能力，那高效微调会更加合适。尽管在2020年前后，深度学习领域诞生了很多高效微调的方法，但现在适用于大模型的最主流的高效微调方法只有一种——LoRA。

LoRA（ Low-Rank Adaptation）微调是一种参数高效的微调方法，旨在通过引入低秩矩阵来减少微 调时需要调整的参数数量，从而显著降低显存和计算资源的消耗。具体来说，LoRA 微调并不直接调整原始模型的所有参数，而是通过在某些层中插入低秩的适配器（Adapter）层来进行训练。

LoRA的原理：

●在标准微调中，会修改模型的所有权重，而在 LoRA 中，只有某些低秩矩阵（适配器）被训练和调整。这意味着原始模型的参数保持不变，只是通过少量的新参数来调整模型的输出。

●低秩矩阵的引入可以在显存和计算能力有限的情况下，依然有效地对大型预训练模型进行微调，从而让 LoRA 成为显存较小的设备上的理想选择。

举个 ：想象你想教学生们怎样进行快速心算而不去完全打破他们原有的学习方法。你决定只引入一个简化版本的心算技巧，让他们在现有知识的基础上进行少量调整。这就像是把原有的学习方式轻量化处理，只增加所需的少量新知识，而不是重新教授整个数学课程。

LoRA的优势：

1.显存优化： 只需要调整少量的参数（适配器），显著减少了显存需求，适合显存有限的GPU。

2.计算效率： 微调过程中的计算负担也更轻，因为减少了需要调整的参数量。

3.灵活性： 可以与现有的预训练模型轻松结合使用，适用于多种任务，如文本生成、分类、问答等。

而QLoRA（Quantized Low-Rank Adaptation） 则是 LoRA 的一个扩展版本，它结合了 LoRA 的低秩适配器和量化技术。QLoRA 进一步优化了计算效率和存储需求，特别是在极端显存受限的环境下。与 LoRA 不同的是， QLoRA 会将插入的低秩适配器层的部分权重进行量化（通常是量化为INT4或INT8），在保持性能的同时显著降低模型的存储和计算需求。

举个 ：针对学生中一些学习资源（如时间或精力）更加有限的情况，你进一步优化教学方法，不仅简化了学习内容（类似LoRA），同时还用了一些有助于记忆的技巧（比如使用图像或口诀），从而更有效地传授知识。这样，每个学生能在有限时间内学会心算法。在技术上，QLoRA涉及量化（quantization）技术，将模型的一部分权重参数存储在较低精度的数值格式中，以此减少内存使用和计算量，同时结合LoRA的低秩调整，让适应过程更加高效。

QLoRA的优势：

1.在显存非常有限的情况下仍能进行微调。

2.可以处理更大规模的模型。

3.适合用于边缘设备和需要低延迟推理的场景。

1.3 高效微调的应用场景

在实际大模型应用场景中，高效微调主要用于以下四个方面：

1. 对话风格微调：高效微调可以用于根据特定需求调整模型的对话风格。例如，针对客服系统、虚拟助理等场景，模型可以通过微调来适应不同的语气、礼貌程度或回答方式，从而在与用户互动时提供更符合要求的对话体验。通过微调少量的参数（例如对话生成的策略、情感表达等），可以使模型表现出更具针对性和个性化的风格。

2. 知识灌注：知识灌注是指将外部知识或领域特定的信息快速集成到已有的预训练模型中。通过高效微调，模型可以更好地学习新领域的专有知识，而无需重新从头开始训练。例如，对于法律、医疗等专业领域，可以使用少量的标注数据对预训练模型进行微调，帮助模型理解特定行业的术语、规则和知识，进而提升专业领域的问答能力。

3. 推理能力提升：高效微调还可以用于提升大模型的推理能力，尤其是在处理更复杂推理任务时。通过微调，模型能够更加高效地理解长文本、推理隐含信息，或者从数据中提取逻辑关系，进而在多轮推理任务中提供更准确的答案。这种微调方式可以帮助模型在解答复杂问题时，提高推理准确性并减少错误。

4. Agent能力（Function calling & MCP能力）提升：在多任务协作或功能调用场景中，高效微调能够显著提升模型Agent能力，使得模型能够有效地与其他系统进行交互、调用外部API或执行特定MCP任务。通过针对性微调，模型可以学会更精准的功能调用策略、参数解析和操作指令，从而在自动化服务、智能助手或机器人控制等领域表现得更加高效和智能。

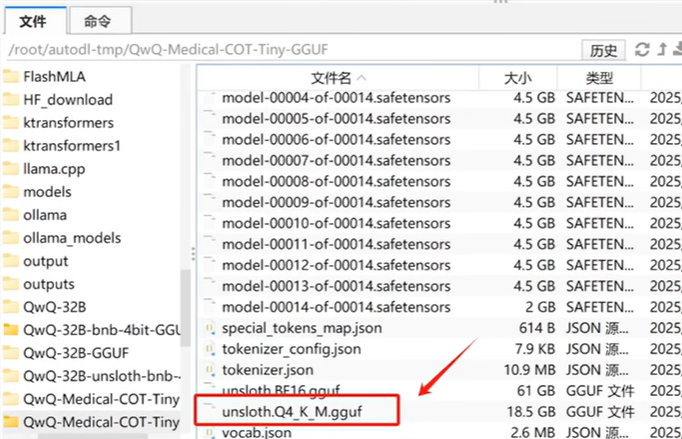
微调源码

|  |
| --- |
| from unsloth import FastLanguageModel  from modelscope.msdatasets import MsDataset  from trl import SFTTrainer  from transformers import TrainingArguments  from unsloth import is\_bfloat16\_supported  train\_prompt\_style = """Below is an instruction that describes a task, paired with an input that provides further context.  Write a response that appropriately completes the request.  Before answering, think carefully about the question and create a step-by-step chain of thoughts to ensure a logical and accurate response.  ### Instruction:  You are a medical expert with advanced knowledge in clinical reasoning, diagnostics, and treatment planning.  Please answer the following medical question.  ### Question:  {}  ### Response:  <think>  {}  </think>  {}"""  max\_seq\_length = 2048  dtype = None  load\_in\_4bit = False  model, tokenizer = FastLanguageModel.from\_pretrained(  model\_name = "./DeepSeek-R1-Distill-Qwen-7B",  max\_seq\_length = max\_seq\_length,  dtype = dtype,  load\_in\_4bit = load\_in\_4bit,  )  EOS\_TOKEN = tokenizer.eos\_token # Must add EOS\_TOKEN  def formatting\_prompts\_func(examples):  inputs = examples["Question"]  cots = examples["Complex\_CoT"]  outputs = examples["Response"]  texts = []  for input, cot, output in zip(inputs, cots, outputs):  text = train\_prompt\_style.format(input, cot, output) + EOS\_TOKEN  texts.append(text)  return {  "text": texts,  }  ds = MsDataset.load('AI-ModelScope/medical-o1-reasoning-SFT', split = "train")  dataset = ds.map(formatting\_prompts\_func, batched = True,)  dataset["text"][0]  model = FastLanguageModel.get\_peft\_model(  model,  r=16,  target\_modules=[  "q\_proj",  "k\_proj",  "v\_proj",  "o\_proj",  "gate\_proj",  "up\_proj",  "down\_proj",  ],  lora\_alpha=16,  lora\_dropout=0,  bias="none",  use\_gradient\_checkpointing="unsloth", # True or "unsloth" for very long context  random\_state=3407,  use\_rslora=False,  loftq\_config=None,  )  trainer = SFTTrainer(  model=model,  tokenizer=tokenizer,  train\_dataset=dataset,  dataset\_text\_field="text",  max\_seq\_length=max\_seq\_length,  dataset\_num\_proc=2,  args=TrainingArguments(  per\_device\_train\_batch\_size=2,  gradient\_accumulation\_steps=4,  num\_train\_epochs = 3,  warmup\_steps=5,  # max\_steps=60,  learning\_rate=2e-4,  fp16=not is\_bfloat16\_supported(),  bf16=is\_bfloat16\_supported(),  logging\_steps=10,  optim="adamw\_8bit",  weight\_decay=0.01,  lr\_scheduler\_type="linear",  seed=3407,  output\_dir="outputs",  ),  )  wandb.init()  trainer\_stats = trainer.train()  new\_model\_local = "DeepSeek-R1-Medical"  model.save\_pretrained(new\_model\_local)  tokenizer.save\_pretrained(new\_model\_local)  model.save\_pretrained\_merged(new\_model\_local, tokenizer, save\_method = "merged\_16bit",) |

来源：https://zhuanlan.zhihu.com/p/30057718169

微调后模型导出到ollama步骤

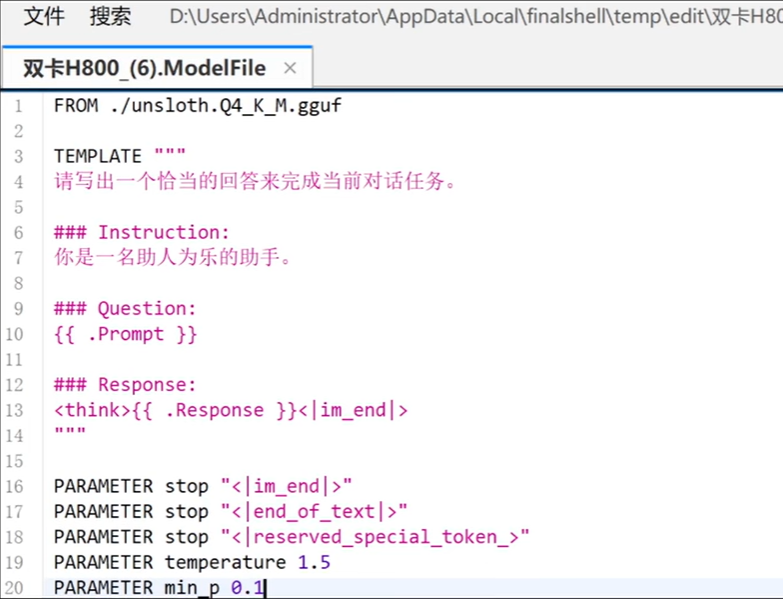
ollama推理  
这里我们需要将创建的Q4\_KM模型权重单独拷贝一份，然后再编写ModelFile文件，便于ollama调用。



然后创建ModelFile:

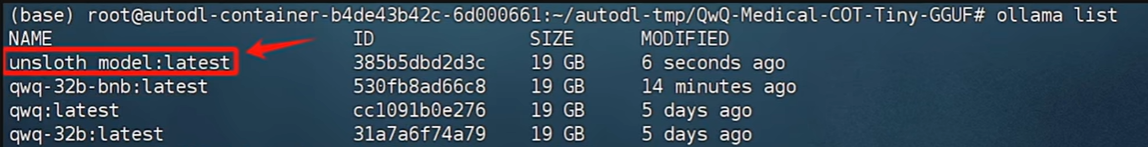


并输入如下内容



保存并退出，然后进行模型注册

ollama create unsloth\_model -f /root/autodl-tmp/QwQ-Medical-COT-Tiny-GGUF/unsloth\_Q4/ModelFile



魔塔蒸馏竞赛冠军训练方案

第一步是用一个蒸馏的方法，用一些思考模型把bird数据集的思维链给它蒸馏出来,接着得到bird nl2sql的思维链数据集。

第二步是对1.5B的基础模型在思维链数据集上做一个简单的监督微调，用了一个比较大的Batch Size,这是为了让训练更加稳定。然后大概在应该是三轮左右就提前终止训练，防止模型过拟合。

第三步用deepseekr1的GRPO算法做一个强化学习，鼓励模型进行自由探索。

https://zhuanlan.zhihu.com/p/27577167893