deepseek复现开源项目

[huggingface/open-r1：DeepSeek-R1 的全开放复刻版](https://github.com/huggingface/open-r1)

deepseek可以做什么

[深度搜索-ai/awesome-deepseek-集成](https://github.com/deepseek-ai/awesome-deepseek-integration)

1.集成web、客户端应用

2.论文阅读器

3.数据分析

4.桌面助手

5.ide插件

6.浏览器插件

7.语音助手

8.代码生成

9.漏洞修复

…

deepseek promot

deepseek promot官方文档

[Prompt Library | DeepSeek API Docs](https://api-docs.deepseek.com/zh-cn/prompt-library/)

deepseek promot特性，promot与其他大模型的差异

[DeepSeek-R1直接把提示工程师的饭碗整没了 - 知乎](https://zhuanlan.zhihu.com/p/20697018627)

[DeepSeek R1: ”你在教我做事？“ - 知乎](https://zhuanlan.zhihu.com/p/20418009560)

(1)使用清晰、简洁且无歧义的提示。

(2)避免对DeepSeek-R1使用few-shot提示，因为这已被证明会在复杂任务中降低性能。相反，使用zero-shot或结构化提示。

Rag

痛点

(1)知识的局限性：模型自身的知识完全源于它的训练数据，而现有的主流大模型（ChatGPT、文心一言、通义千问…）的训练集基本都是构建于网络公开的数据，对于一些实时性的、非公开的或离线的数据是无法获取到的，这部分知识也就无从具备。

(2)幻觉问题：所有的AI模型的底层原理都是基于数学概率，其模型输出实质上是一系列数值运算，大模型也不例外，所以它有时候会一本正经地胡说八道，尤其是在大模型自身不具备某一方面的知识或不擅长的场景。而这种幻觉问题的区分是比较困难的，因为它要求使用者自身具备相应领域的知识。

(3)数据安全性：对于企业来说，数据安全至关重要，没有企业愿意承担数据泄露的风险，将自身的私域数据上传第三方平台进行训练。这也导致完全依赖通用大模型自身能力的应用方案不得不在数据安全和效果方面进行取舍。

rag技术总结：

RAG（中文为检索增强生成） = 检索技术 + LLM 提示。

rag实现过程

完整的RAG应用流程主要包含两个阶段：

数据准备阶段：数据提取——>文本分割——>向量化（embedding）——>数据入库

应用阶段：用户提问——>数据检索（召回）——>注入Prompt——>LLM生成答案

### **优点**

1. **多样化上下文覆盖**
   * 通过多视角生成查询，检索结果涵盖更全面的信息维度，减少单一视角的局限性，提升回答的多样性。
2. **灵活的可控性**
   * 提供多级调节机制（如查询生成数量、权重分配等），便于根据场景需求优化检索与生成效果。
3. **语义纠错与上下文增强**
   * 利用LLM自动修正用户输入的拼写错误，补充上下文（如用户信息、时间等），优化搜索意图理解。
4. **潜在成本优势**
   * 向量检索成本显著低于LLM生成，若额外查询生成能减少后续长文本调用（如避免多次追问），整体成本可能更低。

### **缺点**

1. **延迟增加**
   * 额外LLM调用（生成多查询）及多轮检索延长响应时间，可能影响实时性要求高的场景（如对话系统）。
2. **语义纠错的局限性**
   * 对领域专有术语或新兴概念敏感，若LLM未覆盖相关知识，可能生成无关查询，导致检索质量下降。
3. **成本不确定性**
   * 若多查询生成未有效减少后续调用（如用户仍需多次追问），额外LLM开销可能抵消成本优势。
4. **结果冗余与噪声**
   * 多查询可能引入相似或低质量内容，需依赖后处理（如去重、排序）保障结果精炼性。

rag更多具体实现细节参考:

[一文读懂：大模型RAG（检索增强生成）含高级方法](https://www.zhihu.com/tardis/zm/art/675509396?source_id=1003)

rag demo

微调：

监督微调（SFT）概述：

SFT 是一种改善和定制预训练 LLM 的方法，通过在较小的数据集上重新训练基础模型，使其能够更好地执行指令和回答问题。

SFT 可以提高模型的整体性能，增加新知识，或适应特定任务和领域。

SFT 技术：

全微调：对预训练模型的所有参数进行重新训练，效果最佳但需要大量计算资源。因为它修改了整个模型，所以它也是最具有破坏性的方法，可能导致灾难性地忘记以前的技能和知识。

LoRA（低秩适应）：是一种流行的参数高效微调技术。它不是重新训练整个模型，而是冻结权重并在每个目标层引入小适配器（低秩矩阵）。这使得 LoRA 能够训练许多远低于完全微调（小于 1%）的参数，从而减少内存使用和训练时间。这种方法是非破坏性的，因为原始参数被冻结，然后可以随意切换或组合适配器。

QLoRA：是 LoRA 的扩展，可提供更大的内存节省。与标准 LoRA 相比，它提供了高达 33% 的额外内存减少，使其在 GPU 内存受限时特别有用。这种效率的提高是以更长的训练时间为代价的，QLoRA 的训练时间通常比常规 LoRA 多 39% 左右。

[FAQ + Is Fine-tuning Right For Me? | Unsloth Documentation](https://docs.unsloth.ai/get-started/beginner-start-here/faq-+-is-fine-tuning-right-for-me" \l "does-fine-tuning-add-new-knowledge-to-my-model)

**大模型微调的优缺点总结**

### **优点**

1. **知识内化与任务专精**
   * 将外部知识直接嵌入模型，使其独立执行特定任务（如小众问题回答、文档总结），无需依赖外部系统。
   * 针对结构化、重复或细微差别的任务，提供高精度响应，超越RAG的泛化能力。
2. **高效推理与独立性**
   * 无需检索步骤，响应速度更快，适用于实时性要求高的场景。
   * 不依赖外部知识库，确保系统在检索故障或知识缺失时仍能稳定运行。
3. **行为与风格定制**
   * 精准控制模型输出风格（如品牌声音、合规要求），增强可控性。
4. **稳健性保障**
   * 作为组合系统的基线，即使RAG检索失败，仍能提供可靠输出。

### **缺点**

1. **灾难性遗忘**
   * 过度微调可能覆盖模型原有通用知识，导致泛化能力下降。
2. **数据与训练成本**
   * 依赖高质量标注数据，训练过程计算资源密集，调整参数复杂度高（如LoRA配置）。
3. **知识时效性受限**
   * 模型知识固化于训练数据，难以动态更新，需重新训练以适配新信息。

### **微调与RAG技术对比**

| ****维度**** | ****微调**** | ****RAG**** |
| --- | --- | --- |
| **知识更新** | 静态（需重新训练） | 动态（依赖实时检索） |
| **任务适应性** | 专精特定任务，高精度 | 泛化性强，依赖外部知识库 |
| **推理速度** | 快（无检索延迟） | 慢（需多步检索与生成） |
| **系统依赖性** | 独立运行 | 依赖检索系统与知识库 |
| **成本** | 前期训练成本高 | 长期检索与LLM调用成本可能更高 |

### **微调与RAG的结合优势**

1. **互补性增强**
   * **核心能力+动态知识**：微调强化任务专精能力，RAG提供实时外部知识，覆盖静态与动态需求。
   * **故障容错**：微调作为回退机制，确保RAG检索失败时系统仍可运行。
2. **效率优化**
   * 微调建立基线能力，减少RAG对复杂检索的依赖；RAG减少微调所需的训练数据量。
3. **精准性与适应性平衡**
   * 微调确保领域内精准响应，RAG扩展知识边界（如处理新兴术语或跨领域问题）。

### **总结**

微调与RAG并非互斥，而是互补技术：

* **微调**适合固化核心知识、要求高精度与速度的场景；
* **RAG**擅长动态知识扩展与多源信息整合；
* **结合方案**可兼顾性能、鲁棒性与成本效益，适用于复杂实际应用（如客服、医疗诊断）。

微调llama代码实例:

[Llama3.2\_(1B\_and\_3B)-Conversational.ipynb - Colab](https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.2_(1B_and_3B)-Conversational.ipynb" \l "scrollTo=FqfebeAdT073)

微调deepseek蒸馏模型实例：

|  |
| --- |
| """ 第1步：加载模型和分词器  使用unsloth优化的FastLanguageModel加载预训练模型 """ from unsloth import FastLanguageModel, is\_bfloat16\_supported from datasets import load\_dataset from trl import SFTTrainer # 用于监督微调的训练器 from transformers import TrainingArguments # 用于配置训练参数  # 模型配置参数 max\_seq\_length = 2048 # 最大序列长度 dtype = None # 数据类型，None表示自动选择 load\_in\_4bit = True # 使用4bit量化加载模型以节省显存 #加载预训练模型和分词器 model, tokenizer = FastLanguageModel.from\_pretrained( model\_name = "deepseek-ai/DeepSeek-R1-Distill-Llama-8B", max\_seq\_length = max\_seq\_length, dtype = dtype, load\_in\_4bit = load\_in\_4bit, ) EOS\_TOKEN = tokenizer.eos\_token # 添加结束符标记  """ 第2步：加载并处理数据集  """  # 格式化提示函数,用于处理数据集中的示例 def formatting\_prompts\_func(examples):  # 从examples中提取问题、思维链和回答  inputs = examples["Question"] # 医学问题列表  cots = examples["Complex\_CoT"] # 思维链列表  outputs = examples["Response"] # 回答列表   # 存储格式化后的文本  texts = []   # 遍历每个示例,将问题、思维链和回答组合成指定格式  for input, cot, output in zip(inputs, cots, outputs):  # 使用train\_prompt\_style模板格式化文本,并添加结束符  text = train\_prompt\_style.format(input, cot, output) + EOS\_TOKEN  texts.append(text)   # 返回格式化后的文本字典  return {  "text": texts,  }  # 加载数据集并应用格式化 dataset = load\_dataset("FreedomIntelligence/medical-o1-reasoning-SFT", "default", split="train[0:50000]",  trust\_remote\_code=True) dataset = dataset.map(formatting\_prompts\_func, batched=True, ) """ 第3步：配置LoRA微调参数  使用LoRA技术进行参数高效微调 """ model = FastLanguageModel.get\_peft\_model(  # 原始模型  model,  # LoRA秩,用于控制低秩矩阵的维度,值越大表示可训练参数越多,模型性能可能更好但训练开销更大  # 建议: 8-32之间  r=16,  # 需要应用LoRA的目标模块列表  target\_modules=[  "q\_proj", "k\_proj", "v\_proj", "o\_proj", # attention相关层  "gate\_proj", "up\_proj", "down\_proj", # FFN相关层  ],  # LoRA缩放因子,用于控制LoRA更新的幅度。值越大，LoRA的更新影响越大。  lora\_alpha=16,  # LoRA层的dropout率,用于防止过拟合,这里设为0表示不使用dropout。  # 如果数据集较小，建议设置0.1左右。  lora\_dropout=0,  # 是否对bias参数进行微调,none表示不微调bias  # none: 不微调偏置参数；  # all: 微调所有参数；  # lora\_only: 只微调LoRA参数。  bias="none",  # 是否使用梯度检查点技术节省显存,使用unsloth优化版本  # 会略微降低训练速度，但可以显著减少显存使用  use\_gradient\_checkpointing="unsloth",  # 随机数种子,用于结果复现  random\_state=3407,  # 是否使用rank-stabilized LoRA,这里不使用  # 会略微降低训练速度，但可以显著减少显存使用  use\_rslora=False,  # LoFTQ配置,这里不使用该量化技术，用于进一步压缩模型大小  loftq\_config=None, )  """ 第4步：配置训练参数和初始化训练器 """  # 初始化SFT训练器 trainer = SFTTrainer(  model=model, # 待训练的模型  tokenizer=tokenizer, # 分词器  train\_dataset=dataset, # 训练数据集  dataset\_text\_field="text", # 数据集字段的名称  max\_seq\_length=max\_seq\_length, # 最大序列长度  dataset\_num\_proc=2, # 数据集处理的并行进程数，提高CPU利用率  args=TrainingArguments(  per\_device\_train\_batch\_size=2, # 每个GPU的训练批次大小  gradient\_accumulation\_steps=4, # 梯度累积步数,用于模拟更大的batch size  warmup\_steps=5, # 预热步数,逐步增加学习率  learning\_rate=2e-4, # 学习率  lr\_scheduler\_type="linear", # 线性学习率调度器  max\_steps=60, # 最大训练步数（一步 = 处理一个batch的数据）  # 根据硬件支持选择训练精度  fp16=not is\_bfloat16\_supported(), # 如果不支持bf16则使用fp16  bf16=is\_bfloat16\_supported(), # 如果支持则使用bf16  logging\_steps=10, # 每10步记录一次日志  optim="adamw\_8bit", # 使用8位AdamW优化器节省显存，几乎不影响训练效果  weight\_decay=0.01, # 权重衰减系数,用于正则化，防止过拟合  seed=3407, # 随机数种子  output\_dir="outputs", # 保存模型检查点和训练日志  ), )   """ 第5步 开始训练 """ trainer.train()   """ 第6步：微调后的模型推理测试 """ question = "一位61岁的女性,长期存在咳嗽或打喷嚏等活动时不自主尿失禁的病史,但夜间无漏尿。她接受了妇科检查和Q-tip测试。基于这些发现,膀胱测压最可能显示她的残余尿量和逼尿肌收缩情况如何?"  #定义提示模板 prompt\_style = """以下是描述任务的指令，以及提供更多上下文的输入。 请写出恰当完成该请求的回答。 在回答之前，请仔细思考问题，并创建一个逐步的思维链，以确保回答合乎逻辑且准确。  ### Instruction: 你是一位在临床推理、诊断和治疗计划方面具有专业知识的医学专家。 请回答以下医学问题。  ### Question: {}  ### Response: <think>{}"""  # 对输入问题进行编码,转换为模型可处理的张量格式并移至GPU inputs = tokenizer([prompt\_style.format(question, "")], return\_tensors="pt").to("cuda")  # 启用模型推理模式,使用Unsloth加速推理速度 FastLanguageModel.for\_inference(model)  # 生成回答 outputs = model.generate(  input\_ids=inputs.input\_ids,  attention\_mask=inputs.attention\_mask,  max\_new\_tokens=1200,  use\_cache=True, )  # 解码模型输出 response = tokenizer.batch\_decode(outputs) print("### 微调后模型推理结果：") print(response[0].split("### Response:")[1])   """ 第7步：保存模型  包括保存完整模型和合并后的模型 """ new\_model\_local = "DeepSeek-R1-Medical-COT-Qwen-1.5B" model.save\_pretrained(new\_model\_local) tokenizer.save\_pretrained(new\_model\_local)  #合并模型并量化成4位gguf保存 model.save\_pretrained\_gguf(new\_model\_local, tokenizer, quantization\_method = "q4\_k\_m") |

加载微调后的模型

|  |
| --- |
| from transformers import TextStreamer from unsloth import FastLanguageModel import torch  model, tokenizer = FastLanguageModel.from\_pretrained(  model\_name = "/home/luke10481/deepseek-ai/DeepSeek-R1-Distill-Llama-8B", # YOUR MODEL YOU USED FOR TRAINING  max\_seq\_length = 2048,  dtype = None,  load\_in\_4bit = True,  device\_map={"": torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")}, ) FastLanguageModel.for\_inference(model) # Enable native 2x faster inference text\_streamer = TextStreamer(tokenizer) prompt\_style = """以下是描述任务的指令，以及提供更多上下文的输入。 请写出恰当完成该请求的回答。 在回答之前，请仔细思考问题，并创建一个逐步的思维链，以确保回答合乎逻辑且准确。  ### Instruction: 你是一位在临床推理、诊断和治疗计划方面具有专业知识的医学专家。 请回答以下医学问题。  ### Question: {}  ### Response: <think>{}"""  # 测试用医学问题 question = "一位61岁的女性,长期存在咳嗽或打喷嚏等活动时不自主尿失禁的病史,但夜间无漏尿。她接受了妇科检查和Q-tip测试。基于这些发现,膀胱测压最可能显示她的残余尿量和逼尿肌收缩情况如何?" inputs = tokenizer([prompt\_style.format(question, "")], return\_tensors="pt").to("cuda")  outputs = model.generate(  input\_ids=inputs.input\_ids,  attention\_mask=inputs.attention\_mask,  max\_new\_tokens=1200,  use\_cache=True, )  # 解码模型输出 response = tokenizer.batch\_decode(outputs) print("### 微调后模型推理结果：") print(response[0].split("### Response:")[1]) |

微调节省显存方式

[Unsloth微调Llama3-8B，提速44.35%，节省42.58%显存，最少仅需7.75GB显存 - 大模型知识库|大模型训练|开箱即用的企业大模型应用平台|智能体开发|53AI](https://www.53ai.com/news/qianyanjishu/1755.html)

但unsloth暂不支持cpu微调，但未来提供

[我可以使用 Unsloth 在 CPU 上微调模型吗？ ·问题 #1106 ·unslothai/unsloth](https://github.com/unslothai/unsloth/issues/1106)

[(99+ 封私信 / 80 条消息) 深度学习训练模型时，GPU显存不够怎么办？ - 知乎](https://www.zhihu.com/question/461811359)

#### **现有显存节省策略**

1. **参数高效微调（LoRA）**
   * **低秩分解**：设置 r=16 控制低秩矩阵维度，减少可训练参数量（仅训练LoRA层参数）。
   * **目标模块选择**：针对注意力层（q\_proj, k\_proj等）和FFN层（gate\_proj等）应用LoRA，聚焦关键模块。
   * **LoRA超参优化**：lora\_alpha=16 平衡更新幅度，use\_rslora=False 避免额外计算开销。
2. **梯度优化技术**
   * **梯度检查点（Gradient Checkpointing）**：启用 use\_gradient\_checkpointing="unsloth"，以时间换空间，减少反向传播时的中间激活值存储。
   * **梯度累积（Gradient Accumulation）**：设置 gradient\_accumulation\_steps=4，降低单卡批次大小（per\_device\_train\_batch\_size=2），缓解显存压力。
3. **混合精度训练**
   * **BF16/FP16支持**：根据硬件支持选择 bf16 或 fp16，降低权重和梯度精度，减少显存占用（BF16动态范围更大，数值稳定性更高）。
4. **8位优化器**
   * **AdamW-8bit**：使用 optim="adamw\_8bit"，量化优化器状态，减少显存占用约50%。

蒸馏：

第三方开源蒸馏deepseek r1项目

[open-thoughts/open-thoughts：Open Thoughts：思维模型的完全开放数据管理](https://github.com/open-thoughts/open-thoughts?tab=readme-ov-file)

蒸馏过程讲解

[使用DeepSeek-R1蒸馏训练自己的本地小模型（Qwen2.5-0.5B），原理流程全讲解，模型数据全给你](https://mp.weixin.qq.com/s/P12FAEFHQ_6aRPHlXKGd-w?token=431651514&lang=zh_CN)

理论部分：

DeepSeek-R1的蒸馏原理非常简单，不像传统的知识蒸馏那样通过教师模型和学生模型的KL散度或交叉熵损失来调整学生模型，其采用了一种更加简单高效的蒸馏方法，即通过DeepSeek-R1产生的数据去微调（SFT）其他模型（学生模型），注入DeepSeek-R1（教师模型）的相关知识。并且DeepSeek声称这种方法得到的小模型效果优于小模型直接进行强化学习。

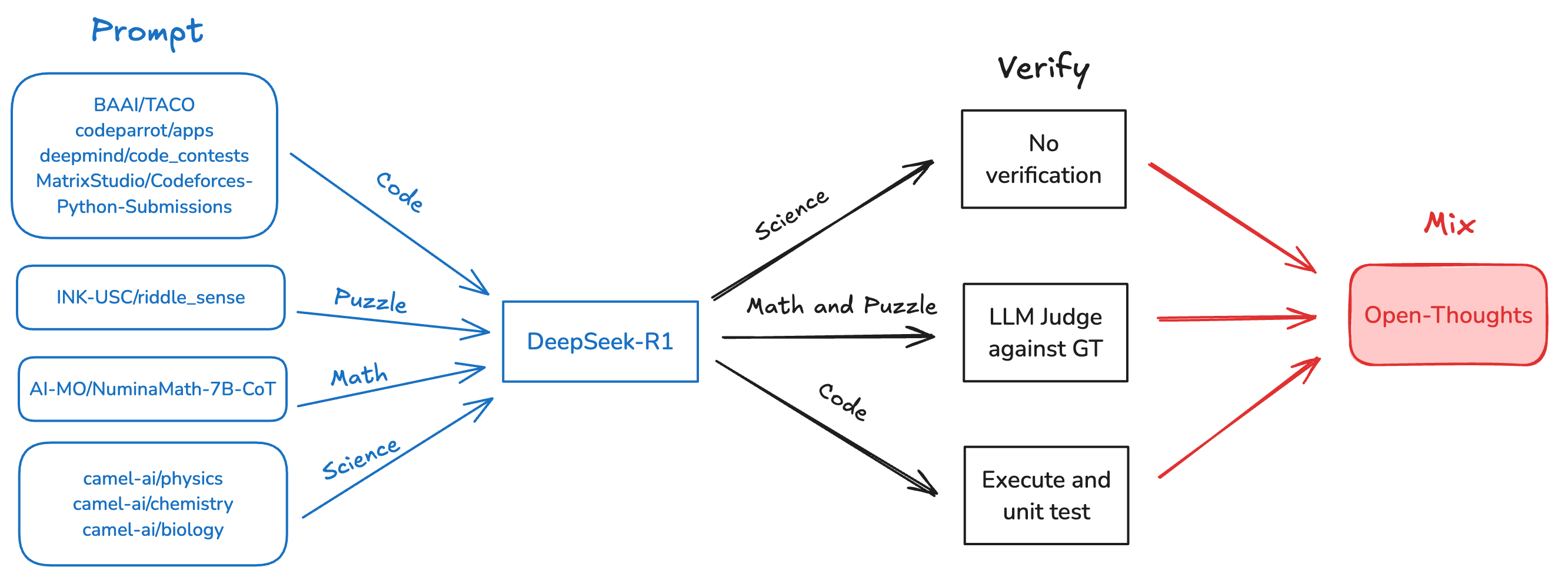
Step1

将不同领域的数据（代码、数学、科学、谜题）输入到DeepSeek-R1模型中，生成带思考过程的答案。

Step2

对生成的结果进行筛选（过滤掉一些不正确的答案），对于科学类的问题（开放性问答），无需验证，对于数学和谜题类的问题，通过大模型对生成结果和标准答案进行评判，对于代码类的问题，通过执行代码通过与否进行判断。

Step3

将所有数据进行混合，得到最终的数据集。

R1 推理模型训练过程：

[在本地训练您自己的 R1 推理模型 （GRPO）](https://unsloth.ai/blog/r1-reasoning)

[Unsloth训练自己的R1推理模型 - DeepSeek GRPO](https://www.youtube.com/watch?v=aNisiIo1xBY)

将 Llama 3.1 （8B） 转换为推理模型：

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

量化：

痛点：

随着语言模型规模的不断增大，其训练的难度和成本已成为共识。 而随着用户数量的增加，模型推理的成本也在不断攀升，甚至可能成为限制模型部署的首要因素。 因此，我们需要对模型进行压缩以加速推理过程，而模型量化是其中一种有效的方法。

概述

大语言模型的参数通常以高精度浮点数存储，这导致模型推理需要大量计算资源。 量化技术通过将高精度数据类型存储的参数转换为低精度数据类型存储， 可以在不改变模型参数量和架构的前提下加速推理过程。这种方法使得模型的部署更加经济高效，也更具可行性。

[source](https://www.cerebras.net/machine-learning/to-bfloat-or-not-to-bfloat-that-is-the-question/)[¶](https://llamafactory.readthedocs.io/zh-cn/latest/advanced/quantization.html" \l "id2)

浮点数一般由3部分组成：符号位、指数位和尾数位。指数位越大，可表示的数字范围越大。尾数位越大、数字的精度越高。

量化可以根据何时量化分为：后训练量化和训练感知量化，也可以根据量化参数的确定方式分为：静态量化和动态量化。

deepseek未量化前671b，文件体积高达 720GB

deepseek完整版1.73-bit动态量化后模型 169GB

[Huzderu/deepseek-r1-671b-1.73bit](https://ollama.com/Huzderu/deepseek-r1-671b-1.73bit)

硬件需求

部署此类大模型的主要瓶颈是内存+显存容量，建议配置如下：

DeepSeek-R1-UD-IQ1\_M：内存 + 显存 ≥ 200 GB

除了模型参数占用的内存+显存空间（158 GB ）以外，实际运行时还需额外预留一些内存（显存）空间用于上下文缓存。预留的空间越大，支持的上下文窗口也越大。

[完整的671B MoE DeepSeek R1怎么塞进本地化部署?详尽教程大放送|内存|gpu|上下文\_网易订阅](https://www.163.com/dy/article/JNDEHFII0511AQHO.html)

低成本部署deepseek r1方案

[(99+ 封私信 / 80 条消息) 如何评价Ktransformers 支持单卡4090推理全量deepseek-R1模型？ - 知乎](https://www.zhihu.com/question/12061570517)

在AI技术飞速发展的今天，大语言模型（LLM）的应用越来越广泛。然而，本地运行这些模型，尤其是像[DeepSeek-R1](https://zhida.zhihu.com/search?content_id=253878480&content_type=Article&match_order=1&q=DeepSeek-R1&zhida_source=entity)这样的大模型，往往需要高性能的硬件支持，这让许多开发者和研究人员望而却步。

今天，为大家推荐一款由清华大学 [MADSys](https://zhida.zhihu.com/search?content_id=253878480&content_type=Article&match_order=1&q=MADSys&zhida_source=entity) 和 [Approaching.AI](https://zhida.zhihu.com/search?content_id=253878480&content_type=Article&match_order=1&q=Approaching.AI&zhida_source=entity) 专为优化大模型本地推理体验而设计的开源框架--[KTransformers](https://zhida.zhihu.com/search?content_id=253878480&content_type=Article&match_order=1&q=KTransformers&zhida_source=entity)。它支持在单卡24GB VRAM的GPU上运行满血版的DeepSeek-R1，较llama.cpp而言，预填充阶段性能提升高达27.79倍！其主要的优化策略有：

1. **混合推理**：KTransformers 框架采用 CPU 和 GPU 混合推理技术。计算密集型操作被卸载到 GPU，而其他操作则由 CPU 处理。这种分工协作的方式能够充分利用硬件资源，提高效率。
2. **专家选择策略**：框架使用了一种基于离线剖析结果的专家选择策略。在推理过程中，选择较少的专家参与计算，在不影响输出质量的前提下，有效地减少了内存占用。
3. **Intel AMX 优化**：框架中使用了 AMX 加速内核和缓存友好的内存布局。这些优化措施显著提升了性能，并减少了内存开销。
4. **高效内存管理**：为了避免 [NUMA 节点](https://zhida.zhihu.com/search?content_id=253878480&content_type=Article&match_order=1&q=NUMA+节点&zhida_source=entity)之间的数据传输成本，框架将关键矩阵复制到两个 NUMA 节点，从而加快预填充和解码过程。虽然这种方法增加了内存消耗，但显著提升了性能。

llm无代码训练框架

[训练方法 - LLaMA Factory](https://llamafactory.readthedocs.io/zh-cn/latest/advanced/trainers.html)

LLaMA Factory 是一个简单易用且高效的大型语言模型（Large Language Model）训练与微调平台。通过 LLaMA Factory，可以在无需编写任何代码的前提下，在本地完成上百种预训练模型的微调，框架特性包括：

* 模型种类：LLaMA、LLaVA、Mistral、Mixtral-MoE、Qwen、Yi、Gemma、Baichuan、ChatGLM、Phi 等等。
* 训练算法：（增量）预训练、（多模态）指令监督微调、奖励模型训练、PPO 训练、DPO 训练、KTO 训练、ORPO 训练等等。
* 运算精度：16 比特全参数微调、冻结微调、LoRA 微调和基于 AQLM/AWQ/GPTQ/LLM.int8/HQQ/EETQ 的 2/3/4/5/6/8 比特 QLoRA 微调。
* 优化算法：GaLore、BAdam、DoRA、LongLoRA、LLaMA Pro、Mixture-of-Depths、LoRA+、LoftQ 和 PiSSA。
* 加速算子：FlashAttention-2 和 Unsloth。
* 推理引擎：Transformers 和 vLLM。
* 实验面板：LlamaBoard、TensorBoard、Wandb、MLflow 等等。

部署推理模型：

大模型部署推理框架对比

ollama部署推理支持cpu+gpu推理方式，它将尝试将尽可能多的 LLM 层卸载到 GPU 上，然后如果它不适合，其他层将在 CPU 上运行。

[Ollama 可以同时使用 CPU 和 GPU 进行推理吗？ ·问题 #3509 ·OLLAMA/OLLAMA](https://github.com/ollama/ollama/issues/3509)

deepseek运行温度参数参考

来源：[unsloth/DeepSeek-R1-Distill-Qwen-32B-bnb-4bit ·拥抱脸](https://huggingface.co/unsloth/DeepSeek-R1-Distill-Qwen-32B-bnb-4bit)

注意：我们建议在运行这些模型时设置适当的温度（介于 0.5 和 0.7 之间），否则您可能会遇到无休止重复或输出不连贯的问题。