【LLM学习】Applied LLMs：LLMs构建应用程序的实践经验总结（1）战术应用

方方 方方的算法花园 2024年06月16日 21:58 浙江

“本文主要针对《What We've Learned From A Year of Building with LLMs》文章进行了翻译和总结，原文是一个非常实用的指南，介绍如何利用LLMs构建成功的产品，文章内容比较长，我会分成战术应用、运营、战略三篇文章进行解读。”

原文地址：https://applied-llms.org/（发布时间：June 8, 2024）

作者：Eugene Yan、Bryan Bischof、Charles Frye、Hamel Husain、Jason Liu、Shreya Shankar

大语言模型技术栈应用的三个层面

1. 战术层面（Tactical）：这一层面关注于使用LLMs构建应用的具体实践，如提示（prompt）设计、检索增强生成（RAG）、流程工程（flow engineering）、评估和监控。它为从业者和短期创新项目提供实操指导，帮助他们改进应用的质量与可靠性。

2. 运营层面（Operational）：运营层面涉及产品推出的日常管理和团队构建，提供高效团队运作的实践指南。这一部分适用于希望可持续、可靠部署产品的技术领导者。

3. 战略层面（Strategic）：战略层面提供长期和宏观的视角，包括关键观点如“在产品市场契合之前不要购买GPU”和“专注于系统而非模型”。它讨论如何进行有效迭代，主要面向创始人和高管。

本文专注于战术应用，深入探讨大语言模型的实操细节。原文作者分享了增强LLM技术栈核心组件的最佳实践，包括：

- 改进提示技巧以提升输出质量和可靠性。
- 实施输出评估策略以确保结果的准确性。
- 利用检索增强的生成技术来加强数据的确凿性。
- 设计涉及人的回路工作流，提升整体流程的效率。

01 Prompt技巧

作者建议在涉及新应用原型时从提示（prompting）开始。人们很容易低估或高估它的重要性。之所以会低估，是因为正确的prompt技巧，如果使用得当，可以让我们走得很远。之所以会高估，是因为即使是基于prompt的应用也需要围绕提示进行大量的工程工作才能良好运作。

1.1 充分发挥基本prompt技巧的效用

（1）n-shot提示：n-shot提示通过提供示例实现上下文学习，帮助LLM理解任务并生成符合预期的输出。

- **示例数量：** 避免使用太少的示例（n过低），以免模型过度依赖特定例子，通常目标是n ≥ 5，甚至可以尝试更多。
- **样本分布：** 示例应代表实际使用时的样本分布，如构建电影摘要生成器时，应包含不同类型电影的样本。
- **示例形式：** 不一定需要提供完整的输入输出对，有时仅提供期望输出的示例即可。

（2）思维链（Chain-of-Thought, CoT）： CoT鼓励LLM在给出答案前展示其思考过程，类似于提供一个草图板。

- **实施方法：** 初始方法包括添加“让我们一步一步地思考”的指令，更具体的说明可以降低幻觉率。
- **应用示例：** 如在总结会议记录时，可以明确列出关键点并检查一致性，最后综合成摘要。
- **研究争议：** 尽管CoT技术的有效性存在争议，但值得尝试。

（3）提供相关资源：提供相关资源可以扩大模型的知识库，减少幻觉，增加用户信任度。

- **检索增强生成（RAG）：** 通过RAG技术为模型提供可直接使用的文本片段。
- **资源使用指导：** 明确告诉模型优先使用这些资源，引用它们，并在资源不足时指出，帮助模型“锚定”到指定资源语料库中。

1.2 结构化输入和输出

结构化输入有助于模型更清晰地理解上下文，明确各个标记之间的关系及其附加元数据，从而提高输出的准确性。例如，在编写SQL的问题中，有效的Text-to-SQL提示应包括结构化的架构定义，以便模型更好地理解任务。结构化输出简化了与下游系统的集成，提高了输出的可用性。

- 对于使用LLM API SDK的情况，推荐使用Instructor工具来实现结构化输出。
- 对于使用Huggingface自托管模型的情况，推荐使用Outlines工具。

每个LLM系列对结构化格式的偏好不同。例如，Claude更喜欢XML格式，而GPT则偏爱Markdown和JSON。使用XML时，你甚至可以通过提供`<response>`标签来预填Claude的响应，如下所示：

```
messages=[
    {
"role": "user",
"content": """Extract the <name>, <size>, <price>, and <color> from this proc
            <description>The SmartHome Mini is a compact smart home assistant
            </description>"""
    },
    {
"role": "assistant",
"content": "<response><name>"
    }
]
```

1.3 使用小型prompts，每个prompt只做好一件事

在软件工程中，避免“万能对象”是一个常见原则。类似地，在构建LLM提示时，也应该避免创建一个做所有事情的单一提示。提示通常从简单开始，但随着添加更多指导和示例以处理边缘情况，复杂性逐渐增加。这可能导致提示变得庞大而难以管理，甚至在处理常见输入时表现不佳。GoDaddy分享了在构建LLM时遇到的这一挑战，说明了过于复杂的提示可能带来的问题（No. 1 lesson from building with LLMs)。

为了保持系统的简洁性，应该将复杂的提示分解为多个简单、专注的步骤：

- **提取关键信息：** 将关键决策、行动项目和所有者提取到结构化格式中。
- **一致性检查：** 检查提取的细节是否与原始记录一致。
- **生成摘要：** 从结构化细节生成简洁的摘要。

通过将单一提示分解为多个简单步骤，我们可以分别迭代和评估每个步骤，提高提示的可管理和可维护性。

1.4 精心打造你的上下文token

重新思考你对于需要发送给agent的上下文数量的假设。要像米开朗基罗一样，不是堆砌你的上下文雕塑——而是凿去多余的材料，直到雕塑显现出来。RAG（检索增强生成）是一种流行的整理所有可能相关的信息块的方式，但你在提取必要信息方面做了什么？

我们发现，将发送给模型的最终提示——包括所有的上下文构建、元提示和RAG结果——放在一张空白页面上，仅仅阅读它，可以极大地帮助你重新思考你的上下文。通过这种方法，我们发现了冗余、自相矛盾的语言和糟糕的格式问题。

另一个关键的优化是你的上下文结构。如果你的文档集合表示对人类没有帮助，不要假设它对agent就有好处。仔细考虑如何组织你的上下文，以强调其部分之间的关系，并使提取尽可能简单。

更多的提示基础，如prompt心智模型、预填充、上下文定位等，可以参考《Prompting Fundamentals and How to Apply them Effectively》（https://eugeneyan.com/writing/prompting/）。

02 信息检索 / RAG

除了prompt之外，引导LLM的另一种有效方法是通过在prompt中提供知识。这使得LLM能够基于所提供的上下文进行上下文学习，这被称为检索增强生成（Retrieval-Augmented Generation，RAG）。实践者发现，与微调（finetuning）相比，RAG在提供知识和改善输出方面非常有效，同时所需的努力和成本要少得多。

2.1 RAG的好坏取决于检索文档的相关性、信息密度和详尽程度

RAG输出的质量取决于检索到的文档的质量，而文档质量又可以从几个因素来考虑：

（1）**相关性**：通常通过排名指标来量化，如平均倒数排名（Mean Reciprocal Rank, MRR）或归一化折扣累积增益（Normalized Discounted Cumulative Gain, NDCG）。MRR评估系统将第一个相关结果排在排名列表中的能力，而NDCG考虑所有结果的相关性及其位置。例如，如果我们正在检索用户评论以生成电影评论摘要，我们将希望将特定电影的评论排名更高，同时排除其他电影的评论。

（2）**信息密度**：如果两个文档具有相同的相关性，我们应该倾向于选择更简洁、包含较少无关细节的那一个。回到我们的电影例子，我们可能认为电影剧本和所有用户评论在广义上都是相关的。尽管如此，评分最高的评论可能在信息上更为密集。

（3）**细节程度**：想象我们正在构建一个从自然语言生成SQL查询的RAG系统。我们可以只提供带有列名的表架构作为上下文。但是，如果我们包括列描述和一些代表性值呢？这些额外的细节可能有助于LLM更好地理解表的语义，从而生成更正确的SQL。

2.2 不要忘记关键词搜索；将其作为基线并在混合搜索中使用

鉴于基于embedding的RAG演示非常普遍，人们很容易忘记或忽视在信息检索领域数十年的研究和解决方案。

尽管embedding无疑是一个强大的工具，但它们并非万能的。首先，虽然它们在捕获高层次的语义相似性方面表现出色，但它们可能难以处理更具体的基于关键词的查询，比如用户搜索名字（例如，Ilya）、缩写（例如，RAG）或ID（例如，claude-3-sonnet）。基于关键词的搜索，如BM25，就显得尤为重要。最后，经过多年的基于关键词的搜索，用户可能已经认为它是理所当然的，如果他们期望检索到的文档没有被返回，可能会感到失望。

“向量嵌入并不能神奇地解决搜索问题。事实上，重要的工作发生在利用语义相似性搜索重新排序之前。真正超越BM25或全文搜索是很难的。”——Aravind Srinivas，Perplexity.ai的CEO

“数月来，我们一直在向客户和合作伙伴传达这一点。使用简单的embedding进行最近邻搜索会产生非常混乱的结果，开始时采用基于关键词的方法通常更为稳妥。”——Beyang Liu，Sourcegraph的CTO

其次，关键词搜索让我们能直观理解文档为何被检索到。相比之下，基于embedding的检索不太可解释。最后，多亏了像Lucene和OpenSearch这样的系统，它们经过数十年的优化和实战检验，关键词搜索通常计算效率更高。

在大多数情况下，混合方法效果最佳：关键词匹配用于明显匹配，emebdding用于同义词、上下位词、拼写错误，以及多模态（例如，图像和文本）。Shortwave分享了他们如何构建其RAG流程，包括查询重写、关键词+emebdding检索和排名（https://www.shortwave.com/blog/deep-dive-into-worlds-smartest-email-ai/）。

2.3 优先使用 RAG 而非微调以获取新知识

RAG和微调都可以用来将新信息整合到LLMs中，并提高特定任务的性能。然而，我们应该优先考虑哪一个呢？最近的研究显示，RAG可能具有优势。一项研究（Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs）比较了RAG和无监督微调（即继续预训练），在MMLU的一个子集和当前事件上评估了两者。他们发现，无论是在训练期间遇到的知识还是完全新的知识，RAG一贯优于微调。在另一篇论文（RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture）中，他们在一个农业数据集上比较了RAG和监督微调。同样，RAG的性能提升大于微调，尤其是对于GPT-4来说。

除了提高性能外，RAG还有其他实际优势。首先，与持续的预训练或微调相比，保持检索索引的最新状态更容易——也更便宜！其次，如果我们的检索索引包含有毒或有偏见内容的问题文档，我们可以轻松地丢弃或修改这些问题文档。

此外，RAG中的“检索”（R）为我们检索文档提供了更细粒度的控制。例如，如果我们为多个组织托管一个RAG系统，通过划分检索索引，我们可以确保每个组织只能检索到它们自己索引中的文档，避免了将一个组织的信息不小心暴露给另一个组织。

2.4 长上下文模型不会使RAG过时

随着Gemin 1.5提供高达1000万个token的上下文窗口大小，一些人开始质疑RAG（检索增强生成）的未来。

“我倾向于认为，Sora对Gemin 1.5的炒作过度了。一个1000万个token的上下文窗口实际上使得大多数现有的RAG框架变得不再必要——你只需将任何数据放入上下文中，然后像平常一样与模型对话。想象一下这对所有将大部分工程努力投入到RAG的初创公司/agent/langchain项目会有什么影响。或者用一句话来说：1000万上下文终结了RAG。干得漂亮，Gemin” —— Yao Fu

虽然长上下文将为分析多个文档或与PDF聊天等场景带来变革，但关于RAG末日的传言显然被极大地夸大了。首先，即使上下文大小达到1000万个token，我们仍然需要一种方法来选择相关的上下文。第二，除了狭窄的“针海觅针”评估之外，我们还没有看到令人信服的数据，证明模型能够有效地处理大上下文尺寸的推理。因此，如果没有良好的检索（和排名），我们冒着用干扰因素压倒模型的风险，或者甚至可能用完全无关的信息填满了上下文窗口。最后，还有成本问题。在推理过程中，Transformer的时间复杂度与上下文长度成线性关系。仅仅因为存在一个模型可以在回答每个问题之前阅读你组织的全部Google Drive内容，并不意味着这是一个明智之举。考虑一个类比：我们如何使用RAM：尽管存在具有数十TB RAM的计算实例，我们仍然需要从磁盘读取和写入。

所以，不要急于把RAG扔进垃圾堆。即使上下文尺寸增长，这种模式仍将保持其用武之地。

03 调整和优化工作流程

仅仅使用LLM prompt只是开始，要充分利用LLM，我们需要考虑整个工作流程。例如，我们如何将一个复杂的任务拆分成多个更简单的任务？什么时候进行微调或缓存有助于提高性能并减少延迟/成本？在这里，我们分享经过验证的策略和现实世界的例子，帮助您优化并构建可靠的LLM工作流程。

3.1 分步骤、多轮次的“流程”可以带来巨大的提升

AlphaCodium通过将单一提示转换为多步骤工作流程，在CodeContests上显著提高了GPT-4的准确率，从19%提升至44%。该工作流程包括：

- 反思问题
- 对公开测试进行推理
- 生成可能的解决方案
- 对解决方案进行排名
- 生成模拟测试
- 在公开和模拟测试上迭代解决方案

具有清晰目标的小任务构成了最佳的agent或流程提示。这有助于提高任务的准确性和效率。虽然并非每个agent prompt都需要请求结构化输出，但在与系统接口协调agent与环境交互时，结构化输出非常有用。

提升性能的策略：

- **明确的计划步骤：** 从预定义的计划中选择，确保步骤的严格性和明确性。
- **重写用户输入：** 将原始用户输入转换为agent prompt，注意可能的信息损失。
- **行为建模：** 将agent行为建模为线性链、有向无环图(DAGs)或状态机，根据任务规模和逻辑关系选择最合适的模型。
- **计划验证：** 包括评估其他agent响应的指令，确保最终组装能够协同工作。

使用固定上游状态的prompt工程，确保你的agent prompt能够针对各种可能情景进行有效评估。

3.2 优先考虑确定性工作流程

尽管AI agent能够动态响应用户请求和环境，但它们的非确定性本质使得部署成为一个挑战。agent采取的每一步都有可能失败，而且从错误中恢复的机会很小。因此，agent成功完成多步骤任务的可能性随着步骤数量的增加而呈指数级下降。这使得开发团队发现很难部署可靠的agent。

一种可行的方法是让agent系统生成确定性计划，然后以结构化、可复现的方式执行这些计划。首先，给定一个高层次的目标或prompt，agent生成一个计划。然后，以确定性方式执行该计划。这使得每一步都更加可预测和可靠。好处包括：

- 制定的计划可用作引导或微调智能体的少样本示例。
- 确定性执行使系统更加可靠，因此更容易测试和调试。此外，失败可以追溯到计划中的具体步骤。
- 生成的计划可以表示为有向无环图（DAGs），相对于静态prompt来说，更易于理解和适应新的场景。

最有经验的agent开发者可能是那些擅长管理初级工程师的专家，因为生成计划的过程类似于我们如何指导和管理初级员工。我们给初级员工明确的目标和具体的计划，而不是模糊的开放式指导，我们也应该对我们的agent这样做。

最终，可靠、有效agent的关键可能在于采用更多结构化、确定性的方法，以及收集数据来优化prompt和微调模型。如果没有这些措施，我们构建的agent可能偶尔能够表现出色，但平均来看，可能会让用户感到失望。

3.3 获得超出温度参数的更多样的输出

在开发推荐商品的大语言模型流程时，如果发现推荐结果过于相似，可以通过增加温度参数来提高输出的多样性。提高温度会使词汇选择的概率分布更均匀，从而增加不常见词汇的出现概率。然而，过高的温度可能导致合适的商品不被推荐，甚至生成无意义的内容。温度调整并不能保证按照期望的概率分布生成结果。

提高多样性的其他方法：

- **调整提示内容：** 改变提示中商品列表的顺序可以显著提升结果的多样性。
- **保持简短的结果列表：** 指示LLM避免推荐最近列表中的物品，或拒绝与最近推荐相似的输出，可以进一步提高多样性。
- **变换措辞：** 在提示中加入不同的短语，如“选择用户会喜欢经常使用的物品”或“选择用户可能会推荐给朋友的产品”，可以转移焦点，影响推荐产品的种类。

3.4 缓存的重要性被低估了

缓存可以通过避免对相同输入的重复计算来节约计算成本并减少响应时间。此外，如果一个响应以前已经被初步审核，那么我们可以直接提供这些经过审核的响应，降低提供有

害或不适当内容的风险。

一种缓存的简单方法是为处理的项目分配唯一ID，比如总结新文章或产品评论时，当一个请求进来时，我们可以检查缓存中是否已经存在摘要。如果存在，我们可以立即返回它；如果没有，我们生成、审查并提供这一内容，然后将其存储在缓存中以供未来的请求使用。

对于更开放式的查询，我们可以借鉴搜索领域的技术，该领域也利用缓存来处理开放式输入。像自动补全、拼写纠正和建议查询等功能也有助于规范化用户输入，从而提高缓存命中率。

3.5 何时进行微调

我们可能有一些任务，即使设计再巧妙的prompt也难以满足需求。例如，即使在显著的提示工程之后，我们的系统可能仍然无法返回可靠、高质量的输出。如果是这样，那么可能需要针对特定任务对模型进行微调。

成功的例子包括：

- Honeycomb的自然语言查询助手：最初，“编程手册”是与n-shot示例一起在提示中提供的，用于上下文学习。虽然这种方法效果尚可，但微调模型在特定领域语言的语法和规则上产生了更好的输出。
- Rechat的Lucy：LLM需要生成以非常特定的格式响应，该格式结合了结构化和非结构化数据，以便前端正确渲染。微调对于其一致性工作至关重要。

尽管微调可以非常有效，但它也带来了显著的成本。我们需要标注微调数据，微调和评估模型，并最终自行托管它们。因此，考虑是否值得更高的前期成本。如果prompt已经让你接近90%的目标，那么微调可能不值得投资。然而，如果我们确实决定进行微调，为了降低收集人工注释数据的成本，我们可以在合成数据上进行微调，或者在开源数据上进行启动。

04 评估与监控

评估LLMs，即使对于顶尖的实验室也认为是一个挑战。因为LLMs返回的是开放式输出，而我们设置给它们的任务是多样化的。然而，严格而深思熟虑的评估至关重要——OpenAI的技术领导者正在专注从事评估工作并对评估结果提供反馈。评估LLM应用引发了多种定义和简化方法：有时它是单元测试，有时又似乎更侧重于观察性，或者可以视为数据科学的一部分。我们认为这些观点都极具价值。本节中，我们将分享在构建评估与监控流程中的一些重要经验。

4.1 从真实的输入/输出样本创建单元测试

在生产环境中，编写单元测试时，应围绕输入输出样本进行，这些测试应基于至少三个评价标准来设定预期结果。若评价标准少于三个，可能意味着任务定义不够具体或过于宽泛，例如通用聊天机器人的测试。

（1）测试触发机制：单元测试应在任何流程更改时自动触发，无论是编辑提示、通过RAG添加新上下文，还是其他修改。文章(https://hamel.dev/blog/posts/evals/#step-1-write-scoped-tests)提供了一个针对实际用例的测试示例。

（2）定义测试用例：建议从定义特定短语的单元测试开始，这些短语有助于筛选或排除某些响应，确保词汇、项目或句子的数量符合预期范围。

（3）单元测试的设定：单元测试应明确指定要包含或排除的响应短语。此外，可以检查生成内容的单词、项目或句子计数是否在合理范围内。对于不同类型的内容生成，断言的形式可能有所不同。

（4）基于执行的评估：执行生成的代码并检查其运行时状态是否满足用户请求，是评估代码生成的一种有效方法。然而，agent代码可能导致运行时状态与目标代码略有差异，因此可能需要对单元测试进行适当的调整。

（5）内部测试：按照客户预期的方式使用产品，即进行"内部测试"，可以揭示真实世界数据的潜在失败模式。这种方法不仅有助于识别潜在弱点，还可以提供实际的生产样本，这些样本可以用于进一步的评估和测试。

4.2 大语言模型作为裁判：有其效用，但非完美解决方案

将LLM作为裁判来评价其他LLM的输出，虽然存在争议，但当这种方法得到恰当实施时，可以与人类评判保持一致性，并帮助我们初步评估新提示或技术的表现。

实施建议：

- **成对比较：** 采用成对比较方式，展示两个选项让LLM选择较优者，这种方法通常能产生更稳定的评估结果。
- **控制顺序偏见：** 为减少选项呈现顺序对LLM决策的影响，每次比较都应交换选项顺序，并确保记录每次比较的胜者。
- **允许平局：** 有时两个选项可能同样优秀，应允许LLM判定为平局，避免不必要的选择。
- **引入思维链（Chain-of-Thought）：** 要求LLM在给出最终判断前先解释其决策过程，这可以提高评估的准确性。此外，这种方式允许使用计算能力较弱的LLM，因为评估通常是批量处理，额外的延迟不会造成问题。
- **控制回答长度：** 避免LLM偏好较长回答的偏见，确保比较的两个回答长度相似，以保证评估的公正性。

大语言模型（LLM）作为裁判的一个关键用途是测试新提示策略是否比旧策略表现更优。通过重新使用已收集的生产结果实例，并应用新的提示策略，LLM可以快速评估新策略的效果。

改进以LLM为评判基准的方法包括记录LLM的反应、评判的批评（即思维链教学法）以及最终结果。这些记录将与利益相关者共同回顾，以识别改进空间。经过三次迭代，LLM与人类的一致性显著提升，从68%增至94%。

尽管LLM作为裁判在某些方面表现出色，但它并非万能的解决方案。在语言的细微差别上，即使是最先进的模型也可能无法可靠地进行评估。此外，我们发现传统的分类方法和奖励模型在准确性、成本效益和响应速度方面可能优于以LLM为基准的评估方法。特别是在代码生成领域，直接的执行评估方法可能更为有效。

4.3 "实习生测试"：评估生成物的有效方法

"实习生测试"是我们在评估生成物时采用的一种方法。该测试通过将完整的输入和上下文信息交给一个相关专业的普通大学生，来评估他们是否能完成这项任务以及所需的时间。

1. 知识缺乏情况：如果大语言模型（LLM）因缺乏必要知识而无法完成任务，我们需要考虑如何丰富上下文信息，以提供更全面的背景知识。

2. 任务复杂性问题：如果问题无法仅通过改善上下文来解决，这可能表明任务对当前的LLM来说过于复杂。在这种情况下，我们需要重新评估任务的可行性或寻找更适合的解决方案。

3. 任务完成时间长：如果实习生能完成任务，但耗时较长，我们应考虑简化任务的复杂性。分析任务是否可以分解，并探索是否可以将某些方面模板化，以提高效率。

4. 快速完成任务：如果实习生能迅速完成任务，我们需要深入分析数据，找出模型可能存在的问题。识别失败的模式，并尝试让模型在作出反应前后进行自我解释，这有助于我们理解模型的思维过程。

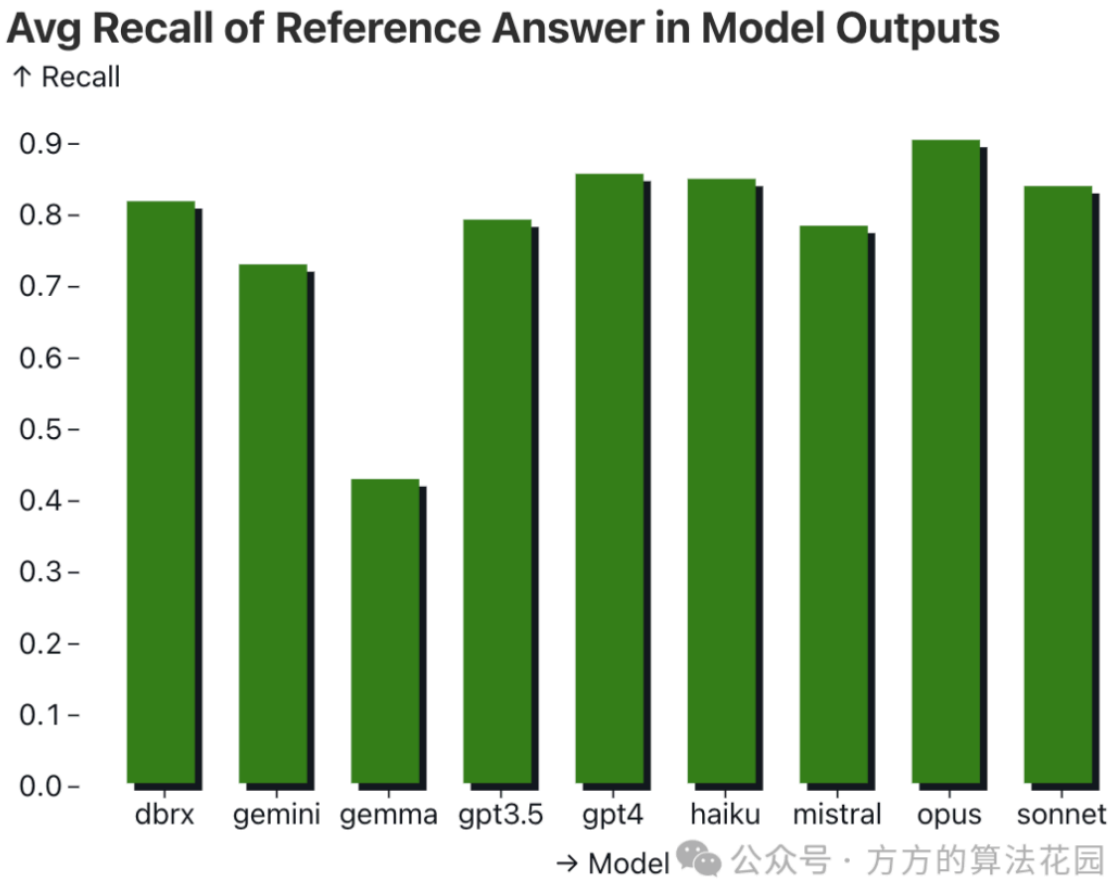
4.4 过度强调某些评估可能会削弱整体表现

古德哈特定律指出，当一项指标变成目标时，它可能不再是一种有效的衡量工具。这在评估大语言模型（LLM）时尤为重要。

“大海捞针 (NIAH)”评估的局限性：NIAH评估最初设计用于量化模型在上下文规模扩大时的回忆能力，以及特定信息位置对回忆效果的影响。然而，随着其在Gemini 1.5报告中被过度强调，我们开始质疑它是否真的能够衡量现实世界中所需的推理和记忆能力。

现实世界任务的挑战：考虑一个更接近实际的场景：如果给定一个小时的会议记录，LLM能否总结出关键决策和行动步骤，并正确归因于相关人员？这种任务不仅涉及机械记忆，还包括解析复杂对话、识别关键信息和合成总结的能力。

NIAH评估的实际应用示例：一个实际应用中的NIAH评估示例是使用医患对话记录，询问LLM关于患者的用药情况。此外，评估还包括了一个更具挑战性的任务：随机插入描述披萨秘密配料的一句话。在药物相关任务中，回忆率约为80%，而在披萨配料任务中，回忆率降至30%。



从某种程度上来说，过度重视 NIAH 评估可能会削弱在信息提取和内容总结任务的表现。由于这些大语言模型 (LLM) 经过精细的微调，专注于分析每个句子，它们可能会错误地把不相关的细节和干扰因素当作重要内容，进而错误地包含在最终的输出中。

这种情况也可能适用于其他评估和应用场景。比如，在进行内容总结时，如果过分强调事实一致性，可能会导致总结过于笼统，降低了事实错误的风险，但相关性可能会受到影响。相反，如果过分追求文风和辞藻的华丽，可能会导致使用过于花哨、具有营销倾向的语言风格，这又可能引入事实上的错误。

4.5 简化为二元任务或对比任务

要求标注者对模型输出进行开放式反馈或在利克特量表上打分，在认知上是一个挑战。由于评价者之间存在差异，收集到的数据通常较为杂乱，影响了数据的实用性。

为了减轻标注者的认知负担并提高数据质量，可以采用以下两种方法：

- **二元分类**：在二元分类任务中，标注者只需对模型输出进行简单的是非判断，如判断摘要的事实一致性、回应的相关性或内容是否包含有害信息。这种方法比利克特量表更精确，评价者间一致性更高，且能显著提高工作效率。例如，Doordash通过一系列是或否的问题来设置菜单项的标签队列。
- **成对比较**：在成对比较中，标注者需要比较两个模型回应并选择较优的一个。这种方法比单个选项打分更直观快捷，能获得更稳定可靠的结果。Thomas Scialom在Llama2聚会上确认，与收集监督式微调数据相比，成对比较的速度更快，成本更低。

在Llama2的案例中，成对比较的成本远低于监督式微调数据的收集。监督式微调数据的成本为每个任务3.5美元，而成对比较的成本高达每个任务25美元。

若你正编写标注指南，这里是 Google 和 Bing 搜索提供的一些 示范指南（https://eugeneyan.com/writing/labeling-guidelines/）。

4.6 无需参考的评估与防护功能的互补性

防护功能旨在识别不适当或有害的内容，而评估功能则专注于检测模型输出的质量与准确度。两者在某些情况下可以互为替代。

当评估不依赖于任何"标准"答案，例如人工撰写的回答时，它可以作为一种防护工具。这种评估方法能够仅通过输入的提示和模型的反应来判断输出的质量。

无需参考的评估应用实例：

- **总结评估：** 在总结评估中，我们可以根据输入的文档来评价摘要的事实一致性和相关性。如果摘要在这些评价指标上表现不佳，可以选择不向用户展示，从而将评估作为一种有效的防护措施。
- **翻译评估：** 类似地，无需参考的翻译评估可以在没有人工翻译对照的情况下，评估翻译的质量。这再次展示了评估作为防护工具的潜力。

4.7 大语言模型的不适当输出挑战

操作大语言模型时，一个显著挑战是它们有时会在不适当的情况下产生输出。这种输出可能是无害但无意义的，有时甚至可能产生有害或危险的内容。

不适当输出的示例：

- 在请求提取文档的特定属性或元数据时，LLM可能会错误地自信地返回不存在的数据。
- 如果输入的是非英语文档，模型可能会用其他语言回答，而不是预期的英语。

尝试引导模型给出“不适用”或“未知”的答复并不总是可靠。预测概率虽然能显示某个词出现的可能性，但并不能确保输出内容的准确性或相关性。

提高输出质量的方法：

- **精心设计的提示：** 可以在一定程度上提高输出质量。
- **安全措施：** 结合使用强大的安全措施，如OpenAI的内容审查API，可以检测潜在的危险内容，如仇恨言论、自伤或色情信息。
- **信息检索技术：** 如果系统找不到相关文档，可以直接回答“我不知道”。

这些安全措施通常与具体应用场景无关，可以广泛适用于所有输出。由于API提供商的延迟或内容审查机制的干预，LLM有时可能无法在需要时输出。因此，持续记录输入和输出（或缺少输出）对于系统的调试和监控至关重要。

4.8 虚假信息生成：一个难以根除的问题

与内容安全或个人信息保护问题相比，虚假信息生成问题更为顽固且难以发现。据大语言模型提供商的信息，即使在执行简单任务如内容总结时，将虚假信息的发生率降低到2%以下仍然是一个挑战。

应对虚假信息的策略：

- **生成前的提示工程：** 采用如思维链（Chain of Thought, CoT）等技术，让大语言模型在生成内容前先解释其推理过程，有助于减少错误生成。
- **生成后的信息真实性检验：** 实施信息真实性检验，通过评估内容摘要的真实性来筛选或重生成错误信息。

信息真实性检验的方法：

- **确定性检测：** 在某些情况下，可以确定性地检测到错误信息。例如，在使用检索增强生成（Retrieval-Augmented Generation, RAG）技术时，如果输出内容结构清晰且能够明确资源来源，我们可以手动核实这些资源是否确实来自于输入内容。

#LLM学习 12

#LLM学习 · 目录

下一篇 · 【LLM学习】Applied LLMs：LLMs构建应用程序的实践经验总结（2）运营策略