Context Engineering

2025年8月12日 1 分钟阅读

#Context Engineering #LLM #AI

> *Context Engineering 是大型语言模型（LLM）应用中的系统性技术，旨在通过动态构建、管理和优化输入模型的信息负载（包括指令、记忆、工具输出、外部知识等），提升模型在复杂任务中的性能、稳定性和可靠性。*

大语言模型上下文工程常见问题解答

什么是上下文工程，它与提示工程有何不同?

上下文工程是为大语言模型（LLM）提供完成任务所需的所有必要信息、指令和工具的艺术和科学。它是一种系统级的方法，涉及构建动态系统，以正确的格式向AI提供它所需的一切，以便在各种场景中保持一致的性能。

与此不同的是，提示工程主要关注为单个查询精心设计巧妙的措辞（例如，“一步一步思考”或“你是一个专家...”）。虽然提示工程是上下文工程的一个组成部分，但它不足以应对日益复杂的LLM应用的需求。上下文工程将重点从静态的、孤立的提示转向动态的、结构化的信息流，从而实现更可靠、更全面的AI系统。

为什么上下文工程对现代LLM应用如此重要?

上下文工程至关重要，因为它解决了大型语言模型（LLM）的几个核心限制和挑战，从而实现了从“令人印象深刻的演示”到“生产就绪的系统”的转变。

> **克服模型无状态性**：早期的LLM是无状态的，这意味着它们只知道你在每次单独的交互中告诉它们什么。上下文工程通过维护记忆、用户偏好和外部信息来解决这个问题，从而使模型能够理解更广泛的交互历史。

目录

文章信息

字数

阅读时间

发布时间

更新时间

标签

#Context Engineering #LLM

解决性能限制：LLM面临计算和内存开销、幻觉、对输入变化的敏感性以及缺乏语义深度等问题。上下文工程通过引入外部知识检索（如RAG）、自我优化和更长的思维链推理等技术来增强性能。

实现资源优化：它提供了传统方法的效率替代方案，通过智能内容过滤、上下文感知调整和动态上下文优化来减少令牌消耗和处理开销，同时保持响应质量。

支持复杂应用：对于法律助理、汽车AI、教育技术和医疗保健等现实世界的应用，模型需要访问和合成来自多个系统的数据。上下文工程确保模型拥有执行这些复杂任务所需的全面、实时和结构化上下文。

上下文工程有哪些核心组成部分？

上下文工程建立在三个核心基础组件之上，这些组件共同管理和优化LLM的信息处理：

上下文检索与生成：这包括通过提示工程制定有效的指令和推理框架，从外部来源（如知识库、数据库、API）动态获取相关信息，以及将这些组件编排成连贯的、任务优化的上下文。

上下文处理：此组件负责转换和优化获取的上下文信息，以最大限度地提高其效用。它解决了超长序列上下文的处理（例如，通过像Mamba这样的架构创新或位置插值），启用迭代自我修正和适应机制，并促进多模态、关系和结构化信息的集成。

上下文管理：此功能侧重于LLM内上下文信息的有效组织、存储和利用。它处理有限上下文窗口带来的基本限制，开发复杂的内存层次结构和存储架构，并实施压缩技术以在保持可访问性和连贯性的同时最大化信息密度。

这些组件协同工作，形成一个全面的框架，确保LLM获得准确、相关且格式正确的信息，从而提高其性能和可靠性。

上下文工程如何整合外部知识？

上下文工程通过一系列策略集成外部知识，从而超越模型固有的训练数据。

检索增强生成（RAG）：RAG是主要的整合方法之一。它将模型参数中存储的参数知识与从外部来源检索的非参数信息相结合，如知识库、数据库和文档集合。这使LLM能够访问当前、领域特定的知识，同时保持参数效率。先进的RAG系统包括模块化架构（如FlashRAG）、代理式RAG（LLM代理动态管理检索）、以及图增强RAG（利用结构化知识图谱进行改进的信息访问）。

知识图谱集成与结构化检索：知识图谱（KG）通过将实体和关系转换为数值向量，解决了结构化信息检索问题，使LLM能够进行多跳推理并改进事实准确性。KAPING和KARPA等框架利用KG检索相关事实，而GraphNeural Networks (GNNs) 则捕捉实体间的复杂关系。

工具集成：LLM可以与外部系统（如数据库、API、搜索引擎）集成。上下文工程确保这些工具的输出得到正确格式化和情境化，以便模型有效使用。例如，ReAct和Toolformer等框架使LLM能够通过功能调用与工具交互，从而扩展其解决问题和获取实时信息的能力。

通过这些方法，上下文工程有效地克服了LLM在处理非训练数据方面的限制，提供了动态、实时的知识，从而提高了模型的准确性、可靠性和多功能性。

上下文工程如何处理长时间交互和“记忆”？

上下文工程通过实施复杂的内存系统和管理策略来处理长时间交互和“记忆”，从而使LLM能够超越无状态交互。

内存层次结构：现代LLM内存架构采用复杂的分层设计。这包括短期记忆（例如，通过上下文窗口和键值缓存实现，用于即时会话）和长期记忆（例如，通过外部数据库或专门结构实现，用于在多个交互周期中保留信息）。MemGPT等受操作系统启发的系统实现了在有限上下文窗口（主内存）和外部存储之间分页信息的能力。

短期记忆管理：为了处理实时交互，上下文工程采用了先进的缓存机制（如键值存储）和上下文窗口（如Transformer架构中的滑动窗口）。这确保了模型能够在短时间内访问和利用最近的交互信息。

长期记忆增强代理：这些代理利用短期记忆（用于实时响应和即时上下文感知）和长期记忆（支持对长时间交互的更深入理解和知识应用）。REMEMBEERER等框架使LLM代理能够利用过去的经验，从成功和失败中学习，而无需进行参数微调。

上下文压缩：为了在有限的上下文窗口内管理大量信息，上下文工程采用了各种压缩技术。这包括自动编码器压缩（如ICAE将长上下文压缩到紧凑的内存槽中），以及递归上下文压缩（RCC）和受生物学启发的遗忘机制（如MemoryBank采用遗忘曲线选择性保留或丢弃信息）。

多轮和纵向交互：上下文工程通过持续状态维护和协作模型之间的内部信息同步来支持多轮对话和顺序任务。这包括跟踪用户偏好、历史交互以及随着时间推移不断演变的领域知识，从而实现个性化和连贯的对话体验。

通过这些机制，LLM可以维护跨越广泛交互的连贯状态，从而模拟人类的记忆和适应能力。

多代理系统在上下文工程中扮演什么角色?

在上下文工程中，多代理系统代表了协作智能的巅峰，通过让多个自主代理协调和通信来解决超出单个代理能力范围的复杂问题。

协作与任务分解：多代理系统通过并行处理、信息共享和自适应角色分配来利用集体智能。LLM的集成增强了这些系统在规划、专业化和任务分解方面的能力。一个"管理器"代理可以将复杂问题分解成更小的子任务，并将它们分配给不同的子代理。

通信协议：标准化协议（如MCP、A2A、ACP和ANP）对于实现不同代理生态系统之间的互操作性至关重要。这些协议允许代理共享上下文、传递消息和协调行动。LLM通过自然语言处理增强了代理通信，从而实现了前所未有的上下文敏感度。

编排机制：编排机制是多代理系统中的关键协调基础设施，管理代理选择、上下文分发和交互流控制。这包括先验编排（在执行前分析输入和能力来选择代理）和后验编排（同时将输入分发给多个代理并根据响应评估结果）。

共享上下文的重要性：一个重要的教训是共享上下文在多代理系统中的必要性。当子代理缺乏对整体目标和彼此行动的完整上下文时，它们可能会产生不匹配或不连贯的结果。解决方案涉及在代理之间共享上下文，或避免将需要紧密连贯的任务拆分为多个独立代理。

上下文工程如何解决多模态和结构化数据的挑战?

上下文工程通过先进的检索、处理和管理策略来应对处理多模态和结构化数据的挑战，从而超越了传统LLM的文本限制。

多模态上下文处理：此领域旨在将文本、图像、音频和视频等多种模态的数据整合到LLM的理解中。挑战包括模态偏见（模型偏爱文本）、细粒度推理缺陷（例如，在视频中理解精确的空间或时间关系）、以及统一不同模态以推断其组合含义。上下文工程通过以下方式应对这些问题：

上下文内学习：MMLM（多模态语言模型）可以从提示中的多模态示例中适应新任务。

长上下文处理：开发能够处理冗长多模态序列的架构，例如视频分析中的自适应分层令牌压缩。

新兴应用：MMLM用于预测推理（如根据视觉场景预测人类活动）、视觉问答（VQA）和数字行动规划（根据感官输入）。

关系和结构化上下文：LLM在处理表格、数据库和知识图谱等结构化数据方面面临限制，因为它们的文本输入要求和顺序架构。上下文工程通过以下方式解决这些问题：

知识图谱嵌入和神经集成：将实体和关系转换为数值向量，并通过图神经网络（GNNs）捕捉复杂关系，从而实现多跳推理。GraphToken等方法通过显式表示结构信息来提高性能。

口头化和结构化数据表示：将结构化数据（如知识图谱三元组和表格行）转换为自然语言句子，以便与LLM无缝集成。编程语言表示（如Python用于知识图谱，SQL用于数据库）在复杂推理任务中表现优于自然语言。

性能提升：结构化数据集成通过将响应锚定在可验证的事实中，减少了幻觉，并提高了LLM在医疗保健、科学研究和商业分析等领域的推理能力和事实准确性。

上下文工程未来的研究方向和开放性挑战是什么?

上下文工程是一个快速发展的领域，未来的研究将集中在以下几个关键方向和开放性挑战：

统一理论基础：目前，上下文工程缺乏统一的理论基础来连接各种技术并提供系统的设计指导。未来的研究需要建立数学框架来描述上下文工程的能力、局限性和最佳设计原则，包括上下文分配和信息冗余。

缩放定律和计算效率：LLM在理解能力和生成长篇、事实一致内容方面的显著差距是一个关键挑战。需要开发能够处理超长序列的新一代架构（如State Space Models），同时保持计算效率和推理质量。

多模态集成和表示：将文本、图像、音频和视频等多种模态有效集成仍然是一个挑战。未来的研究需要侧重于跨模态推理、统一架构设计和更抽象信息的处理，例如图结构。

智能上下文组装和优化：开发能够智能地从可用组件中组装上下文的自动化系统是一个关键的研究前沿。这需要上下文优化算法、自适应选择策略和学习组装功能。

领域专业化和适应：针对医疗保健、法律、科学、教育和工程等特定领域定制上下文工程系统。这包括知识集成、推理模式、安全考虑和法规遵从性方面的独特要求。

大规模多代理协作：将多代理上下文工程系统扩展到数千个代理需要开发分布式协调机制、高效的通信协议和分层管理结构。解决跨复杂工作流的事务完整性和容错能力至关重要。

人机协作与集成：需要深入理解人类认知过程、沟通偏好和信任动态，以实现利用互补优势的有效混合团队。研究将集中于优化任务分配、沟通协议和人类与AI系统之间共享心智模型的开发。

部署和社会影响：大规模生产部署要求解决可扩展性（计算资源、延迟、吞吐量、成本效益）、可靠性、维护性、安全性和

鲁棒性方面的挑战。评估框架必须解决偏见缓解、隐私保护和透明度问题，以确保负责任的开发和部署。

这些相互关联的挑战需要持续的、协作的研究，以确保上下文工程系统在日益融入关键社会功能时保持有益、可靠并符合人类价值观。

目录

Claude-Code-Router (CCR) 是一款创新的AI模型智能路由工具，它通过拦截 Claude Code 应用对 Anthropic Claude模型的请求，进行多维度分析（如Token数量、用户指令、任务类型），然后依据动态路由规则和配置，将请求智能地导向最合适的AI模型（来自如Gemini、DeepSeek、本地Ollama模型等不同的模型服务提供商）。CCR的核心机制包括API格式的自动转换与适配、基于Express.js的中间件架构、异步请求处理，以及完善的错误检测、自动降级到兜底模型和潜在的重试策略，旨在提升AI服务调用的效率、灵活性和成本效益。

深入解析 Claude-Code-Router：AI 时代

的智能路由中枢

1. 引言：AI服务智能路由的新范式

在人工智能（AI）技术飞速发展的今天，大语言模型（LLM）已成为推动各行各业变革的核心引擎。然而，随着模型数量的激增以及它们在能力、性能和成本上的显著差异，如何高效、智能地管理和调度这些模型，以最大化其价值并满足多样化的应用需求，成为了一个亟待解决的关键问题。传统的单一模型服务模式已难以适应日益复杂的应用场景，开发者常常需要在不同模型的 API 之间进行繁琐的切换和适配，这不仅增加了开发成本，也限制了应用的整体性能和灵活性。正是在这样的背景下，**Claude-Code-Router (CCR)** 应运而生，它代表了一种全新的 AI 服务智

能路由范式。CCR 通过其精心设计的核心算法与架构，特别是其智能路由决策机制、请求转换与转发策略以及错误处理与降级策略，为多模型的高效协作与按需调度提供了强大的技术支撑。本文将深入探讨 CCR 的这些核心技术，旨在为资深技术专家和架构师提供一个全面而深入的理解，以便更好地评估和应用此类智能路由解决方案，从而在 AI 时代构建更强大、更灵活、更经济的应用系统。

2. Claude-Code-Router 核心机制总览

Claude-Code-Router (CCR) 的核心机制围绕着如何智能地拦截、分析、路由、转换和转发用户请求到最合适的 AI 模型，并将模型的响应有效地返回给用户。这一过程可以概括为一个精细化的处理流水线，确保了请求在整个生命周期中得到高效和准确的处理。CCR 的设计理念在于解耦用户请求与具体模型服务，通过一个中间层来动态管理请求的流向，从而实现模型选择的

灵活性、成本的可控性以及服务的鲁棒性。这个中间层，即 CCR 本身，扮演着 AI 服务智能交通枢纽的角色，根据实时的请求特性和预设的策略，将任务分配给最匹配的模型实例。

2.1. 请求拦截与预处理

CCR 的首要步骤是有效地拦截来自客户端（例如 Claude Code 工具）的 API 请求。这是通过一种巧妙的环境变量劫持机制实现的。具体而言，CCR 利用了 Claude Code 工具本身支持通过环境变量 `ANTHROPIC_BASE_URL` 来覆盖其默认 API 端点地址的特性。通过设置此环境变量，可以将原本直接发送给 Anthropic 官方 API 的请求，重定向到 CCR 本地运行的服务器地址（例如 `http://localhost:3456`）。这种拦截方式无需修改 Claude Code 工具的源代码，实现了对请求流的无侵入式接管，极大地简化了部署和集成过程。一旦请求被成功拦截到 CCR 的本地服务，预处理阶段随即开始。这个阶段主要包括对传入请求的初步

校验、日志记录以及为后续的智能路由决策准备必要的上下文信息。例如，CCR 可能会提取请求头中的关键信息，或者对请求体进行初步解析，以确保请求的完整性和有效性，并为后续的分析步骤提供基础数据。

Llama 4 模型系列

本文介绍了Llama 4 模型系列详...

DeepSeek 开源 LLM ...

本文介绍了DeepSeek 开 ...