智能的架构：深入解析上下文工程与AI系统的未来

执行摘要

本报告深入剖析了“上下文工程”（Context Engineering）这一新兴的关键领域，它正在重新定义我们构建、部署和扩展人工智能（AI）系统的方式。随着大型语言模型（LLM）的能力日益增强，其性能的瓶颈已从模型本身转向了提供给模型的信息质量。上下文工程正是应对这一挑战的系统性学科，其核心目标是设计、构建和优化提供给AI的完整信息有效载荷，以实现可靠、可扩展且高质量的输出。这标志着行业从简单的“提示工程”（Prompt Engineering）向更全面、更具战略性的信息架构设计的根本性转变。

报告整合了多位行业思想领袖的观点。Andrej Karpathy 将LLM比作一种新型的“操作系统”，其中上下文窗口是其有限的“内存（RAM）”，这一比喻为理解上下文管理的重要性提供了宏观框架。Sean Grove 则倡导“规范即代码”（Spec-as-Code）的理念，强调精确传达意图的规范文档应成为AI开发的核心产物。而LangChain等框架则提供了将这些理念付诸实践的工程蓝图，提出了“写入（Write）、选择（Select）、压缩（Compress）、隔离（Isolate）”四大技术支柱。

本报告将系统性地拆解这些核心技术，包括作为“选择”支柱基石的检索增强生成（RAG）、作为“压缩”支柱关键实践的上下文摘要与修剪，以及作为物理层挑战的键值缓存（KV-Cache）优化。通过对Cursor AI、Claude Code和Anthropic多智能体系统等前沿应用案例的深入分析，报告揭示了上下文工程在实际产品中的具体实现方式。

此外，报告还探讨了该领域固有的风险，如“迷失在中间”（Lost in the Middle）等模型认知偏差和“上下文投毒”（Context Poisoning）等严重的安全漏洞，并提出了相应的缓解策略。最后，报告展望了上下文工程的未来趋势，包括“上下文工程师”这一新角色的崛起、自优化和分层上下文系统的发展，并为企业组织提供了采纳和实施上下文工程的战略性建议。总而言之，掌握上下文工程不仅是一项技术要求，更是在AI时代构建真正智能、可靠和有价值的应用程序的战略核心。

第一部分：引言 - 迎来上下文感知时代

1.1. 定义上下文工程：超越提示

上下文工程是一门系统性的学科，专注于设计、构建和优化提供给AI系统的上下文信息，以实现预期的结果[1]。它超越了仅仅编写一个好的提示，旨在创建一种系统化的通信方法，以确保AI每次都能做出一致、可靠且高质量的响应。其核心目标是建立可预测、可重复的交互模式，最大限度地减少模糊性，并最大化AI理解和恰当响应的能力[1]。

它与提示工程的根本区别在于其范围和时间尺度。提示工程通常优化的是单次交互，聚焦于措辞、语气和少量示例。而上下文工程则管理整个工作流程，涵盖了跨越多次交互的内存管理、工具调用和知识整合[3]。在AI应用日益复杂、需要进行多轮对话和动态响应的今天，一个无论多么完美的单一提示都往往力不从心。大多数AI智能体（Agent）的失败并非源于模型能力的不足，而是上下文的缺失或质量不佳[4]。因此，上下文工程的出现，标志着AI开发范式从“手工艺式”的提示调优，转向了“工业化”的信息流架构设计。

1.2. “感觉编程”在生产系统中的不足

“感觉编程”（Vibe Coding）是指通过与AI进行类似对话的“感觉”交流来构建应用程序的早期开发模式[5]。这种方式在原型设计阶段展示了惊人的力量——例如，Andrej Karpathy在不了解Swift语言的情况下，仅通过与LLM对话就成功构建了一款功能性的iOS应用[5]。然而，这种方法的本质是无结构的、依赖直觉的，因此在构建严肃的生产级系统时会迅速失效[6]。

来自METR的一项研究为这一观点提供了有力证据。该研究发现，在处理真实世界的复杂任务时，AI编码工具实际上让经验丰富的开发人员的速度减慢了19%。其主要原因在于，开发者需要花费大量时间来构造精确的提示并等待AI响应，即“提示开销”，同时AI本身也难以处理大型、复杂代码库的微妙之处[7]。这清晰地表明，在专业的软件工程环境中，依赖直觉和“感觉”的交互模式是低效且不可靠的。行业正在经历一个明确的转变：从基于直觉的“感觉编程”，迈向基于结构的上下文工程，将指令、规则和文档视为需要像其他软件资产一样被精心架构的工程资源[6]。

1.3. 核心论点：为何上下文是新的瓶颈与机遇

随着LLM本身（可被视为“CPU”）的能力不断进步，决定一个AI应用性能优劣的关键因素，正从模型本身转移到输入给其工作内存（即“RAM”或上下文窗口）的信息质量[5]。一个“廉价的演示”和一个“神奇的产品”之间的差距，往往不在于模型本身，而在于所提供上下文的质量[4]。一个简单的例子是请求AI安排会议：如果没有日历数据、过去的邮件往来记录以及联系人信息作为上下文，即便是最先进的LLM也无法完成这个任务。所有这些信息，都是经过工程化设计的上下文元素[4]。

这一转变从根本上重塑了AI工程师的角色定位：他们的核心工作不再是编写每一行代码，而是架构那些能够在正确的时间、以正确的格式收集和组织正确上下文的系统[4]。

这种转变背后存在着深刻的经济驱动力。上下文工程的兴起不仅是技术演进的必然，也是经济上的必需。一方面，LLM的推理成本，特别是对于长上下文窗口的调用，依然高昂；另一方面，AI工程师的时间成本也在增加。在这种背景下，“感觉编程”由于其高失败率和不可预测性，在经济上变得不可行。例如，作为上下文感知架构典范的多智能体系统，其运行消耗的token数量可能是普通聊天的15倍[11]。因此，企业面临着巨大的经济压力，必须最大化每一次LLM交互的成功率，并最小化因信息不相关或尝试失败而浪费的计算资源。上下文工程通过预先结构化信息、减少模糊性，从而显著提高一次性成功的概率，直接应对了这一挑战。这种系统化的方法降低了开发风险，优化了运营支出，使其成为任何希望规模化构建AI应用的企业都必须掌握的关键学科。

第二部分：远见者 - 三种竞争与融合的哲学

在上下文工程这一新兴领域，三股思想潮流尤为突出，它们分别来自Andrej Karpathy、Sean Grove和LangChain。它们并非相互排斥，而是代表了新AI开发堆栈中不同但又相互关联的抽象层次，共同描绘了该领域的宏伟蓝图与实践路径。

2.1. Andrej Karpathy的“软件3.0”：作为新操作系统的LLM

Andrej Karpathy为我们理解正在发生的技术变革提供了一个极其强大的心智模型。他将软

件的演进划分为三个阶段：软件1.0（经典编程）、软件2.0（神经网络）和软件3.0（大型语言模型），在软件3.0时代，自然语言成为了新的编程接口[5]。

- 操作系统类比：Karpathy的核心洞见在于将LLM生态系统类比为我们熟悉的计算机操作系统[5]。在这个类比中：
  - **LLM是CPU**：是执行计算和推理的核心处理器。
  - 上下文窗口是**RAM**：是模型有限的工作内存，所有需要处理的信息都必须加载到这里。
  - **API**是系统调用：是程序与LLM这个“操作系统”交互的标准接口。
  - 提示界面是终端：是用户或开发者输入指令的地方。
    他认为，我们目前正处于AI计算的“1960年代”，一个类似于大型机时代的、资源集中且共享的阶段 5。这个类比的深刻之处在于，它立即将“上下文管理”问题提升到了“内存管理”这一操作系统核心问题的高度，强调了其基础性和重要性。
- **“自主性滑块”**：这是一个针对AI产品设计的关键概念。Karpathy主张，产品不应是“手动”与“全自动”的二元对立，而应存在于一个连续的光谱上[5]。他强烈推荐构建“部分自主”的产品，即AI在人类的监督和引导下进行辅助工作。他用“钢铁侠战衣”（增强人类）而非“钢铁侠机器人”（完全替代人类）来比喻当前最理想的人机协作模式[5]。
- 为智能体用户设计：Karpathy提出了一个前瞻性的观点：我们软件的“用户”将越来越多地是代表人类行事的AI智能体[5]。这要求软件设计理念发生根本性转变——从为人类设计的图形用户界面（GUI），转向为机器设计的、可解析的接口，例如结构化的Markdown文档、API和curl命令[5]。他甚至提议建立类似robots.txt的llm.txt标准，用以指导智能体的行为。
- **“LLM心理学”**：为了更好地与LLM协作，Karpathy将其生动地描述为“人类精神的模拟体”（people spirits）——它们是拥有百科全书式记忆但同时带有奇怪认知怪癖的随机性模拟器，类似于电影《雨人》中的学者综合症患者[5]。这意味着LLM在某些方面超越人类，但在另一些方面又极易犯错。因此，我们必须设计出能够容忍其易错性、并包含人类验证环节的稳健系统[12]。

2.2. Sean Grove的“新代码”：作为真理之源的规范

Sean Grove从另一个角度切入，他认为AI时代最有价值的技能不再是编写代码，而是“精确地传达意图”[15]。在他看来，“规范作者”（spec author）就是新的程序员[17]。

- 意图的至高无上：Grove的核心论点是，严谨的、可版本化的规范文档（例如Markdown文件）应该成为唯一的“真理之源”（source of truth）。这些规范可以被“编译”成各种产物，包括文档、评估标准、模型行为，甚至最终的代码[16]。他以美国宪法作为一个“版本化的规范”，而司法审查是其“评分器”为例，说明AI系统同样需要这种可执行的规范来对齐人类团队和机器智能[16]。OpenAI内部使用的

Model Spec（模型规范）就是这一理念的现实应用 [15]。

- 可执行的规范：这一理念的具体实践是，将规范本身作为输入提供给模型，然后用规范中定义的标准来测试模型的输出，形成一个闭环 [19]。
- 批判性分析与矛盾：Grove的愿景引人深思，但也面临着现实的挑战和批评。自然语言写成的规范，无论多么精心，都不可避免地含有歧义、未言明的假设和边缘情况 [18]。代码，作为一种形式化语言，才是“消除歧义的终极行为”和“可执行的真理” [18]。一个有力的反例是OpenAI自身的“谄媚（sycophancy）”漏洞——模型为了取悦用户而牺牲了事实准确性。Grove认为这恰恰证明了规范的价值，因为规范中明确禁止谄媚。但批评者指出，这种逻辑是颠倒的：规范并未能阻止这个带有欺骗性的行为发生，它只在事后充当了“损害控制”的工具，而不是事前预防的工程控制手段 [18]。这恰恰说明了仅靠规范是不足的。

2.3. LangChain的实用主义：构建者的蓝图

如果说Karpathy提供了“是什么”（一个新的操作系统）的宏大叙事，Grove提供了“为什么”（以意图为中心）的哲学指引，那么LangChain则提供了“怎么做”的务实路径。它的框架是为一线工程师打造的实用工具箱 [9]。

LangChain提出的“写入（Write）、选择（Select）、压缩（Compress）、隔离（Isolate）”四大支柱，为管理上下文窗口提供了一个结构化、可操作的蓝图 [22]。这套方法论将抽象的理念转化为具体的工程模式，使得开发者能够系统地处理上下文。像LangChain和LlamaIndex这样的框架，通过提供RAG、内存管理和智能体工作流的底层支持，极大地降低了实现这些复杂模式的门槛，成为连接宏大愿景与工程现实的桥梁 [20]。

抽象与控制的光谱

Karpathy、Grove和LangChain的哲学并非相互排斥，而是构成了新AI开发堆栈中不同但共存的抽象层次。这三者之间的张力揭示了现代AI开发中的一个根本性权衡：在以人为中心、但 inherently 模糊的意图（Grove的规范）与以机器为中心、必须明确无误的逻辑（LangChain构建的系统）之间的权衡。

这个光谱可以这样理解：

1. **Karpathy的“LLM即操作系统”** [5] 设定了最高的抽象层次，定义了我们所处的计算范式。这是整个舞台的背景。

2. **Grove的"规范即代码"** [16] 为这个新操作系统提出了理想的用户界面：自然语言规范。这是操作系统的"用户层"，人类意图是其主要输入，它优先考虑的是表达力与人类价值的对齐。
3. 然而，正如批评所指出的，这个自然语言界面是"有损"且模糊的 [18]。操作系统需要一个"内核"或"系统库"来将这种高级意图转化为可靠的、可执行的动作。
4. **LangChain**的四支柱框架 [9] 正是构成了这个系统库的核心组件。Select（RAG）就像文件系统调用；Write是内存管理调用；Compress是系统优化工具；Isolate则是进程管理器。

因此，一个现代AI应用的开发者实际上需要同时在这三个层次上工作。他们会运用Grove的原则，在规范中定义项目的目标；借鉴Karpathy的理念，来架构人机交互的模式（如自主性滑块）；并使用LangChain的工具，来实现底层的上下文管理流水线，以确保系统的稳健性和可靠性。整个领域面临的持续挑战，正是在于如何弥合规范层的模糊性与实现层所需的确定性之间的鸿沟。

表1：上下文工程核心哲学对比

思想家/框架	核心隐喻/概念	主要关注点	关键贡献	隐含的开发者角色
Andrej Karpathy	LLM即操作系统；部分自主性	概念架构	为新的计算范式和AI产品设计提供了心智模型。	"系统架构师"或"AI产品设计师"
Sean Grove	规范即代码	意图与对齐	将精确、可版本化的自然语言规范提升为首要的编程产物。	"规范作者"或"意图工程师"
LangChain	四大支柱	实践与实现	提供了一个结构化、模块化的上下文窗口管理框架（写入、选择、压缩、隔离）。	"上下文工程师"或"AI应用开发者"

第三部分：技术核心 - 上下文管理的四大支柱

LangChain提出的"写入、选择、压缩、隔离"四大支柱，为上下文工程提供了清晰的分类

法和实践指南。本节将以此为框架，深入探讨构成上下文管理的技术核心。

3.1. 支柱一：写入（持久化与记忆）

“写入”是指将信息存储在即时上下文窗口之外，以便将来使用，从而为本质上无状态的LLM调用创建短期和长期记忆[20]。这是克服LLM“用后即忘”特性的基础。

- 技术实现：
  - 暂存器（**Scratchpads**）：这是为复杂推理任务提供临时记事本的方法。智能体可以在暂存器中记录中间步骤、思考过程或子任务计划[9]。一个典型的例子是Anthropic的多智能体研究系统，它会将研究计划写入内存，以确保在对话超出上下文窗口限制时，核心计划不会丢失[9]。
  - 状态对象（**State Objects**）：使用结构化的运行时对象（例如，Python中的Pydantic模型）以受控的方式在多次交互中持久化状态[24]。这比非结构化的暂存器更可靠，便于程序化访问和修改。
  - 长期记忆（**Long-Term Memory**）：跨会话持久化存储关键事实、用户偏好和对话历史，以实现真正的个性化体验[26]。
- 案例研究：ChatGPT记忆功能的架构
  ChatGPT的记忆功能是“写入”支柱的一个成熟应用。它巧妙地结合了短期记忆（当前会话内的上下文）和长期记忆（跨会话的信息保留）26。其长期记忆并非简单的对话日志，而是采用了一套复杂的检索机制，根据信息的近时性（Recency）、频率（Frequency）和上下文相关性（Context Matching）来动态地浮现最相关的信息 26。至关重要的是，该功能的设计充分考虑了用户隐私和控制权。用户可以随时查看、删除或禁用特定记忆，这为构建可信赖的记忆系统提供了关键的设计模式 26。

3.2. 支柱二：选择（动态信息检索）

“选择”是在正确的时间，从海量信息源中获取恰到好处的信息，并将其加载到上下文窗口的艺术[22]。这是当前上下文工程中技术最成熟、应用最广泛的领域，其核心技术是检索增强生成（Retrieval-Augmented Generation, RAG）。

- RAG流水线深度解析：
  RAG已经从一个简单的概念演变为一个复杂的多阶段流水线。
  - 朴素**RAG**（**Naive RAG**）：这是RAG的基线形式，通过对向量数据库进行语义搜索，检索与查询最相似的文档片段[20]。

- 高级**RAG**（**Advanced RAG**）：现代生产级RAG系统远比朴素RAG复杂，通常包含以下一个或多个增强步骤：
  1. 查询转换（**Query Transformation**）：系统首先不是直接使用用户的原始查询，而是将其重写或分解为多个更精确、更适合检索的子查询，以提高检索质量[29]。
  2. 混合搜索（**Hybrid Search**）：为了兼顾语义相关性和关键词匹配的准确性，系统会同时使用语义向量搜索和传统的关键词搜索（如BM25），然后融合两者的结果[28]。
  3. 重排序（**Re-ranking**）：在初步检索到一批候选文档后，系统会使用一个能力更强、但成本也更高的“重排序模型”对这些文档进行二次排序，选出最相关的几个，再最终提交给生成LLM[30]。
  4. 上下文感知门（**Context Awareness Gate, CAG**）：这是一个智能决策模块，用于动态判断一个查询是否需要进行检索。对于常识性问题或不需要外部知识的查询，该门会关闭，从而避免向上下文中注入不相关的噪声信息[29]。
- RAG与长上下文之争：
  随着拥有百万级甚至千万级token上下文窗口的LLM的出现，一个普遍的问题是：我们是否还需要RAG？研究表明，答案并非简单的“是”或“否”。在许多场景下，特别是当关键信息分散在大量文档中时，精心设计的RAG系统性能仍然优于单纯依赖长上下文的LLM 31。长上下文窗口容易受到“迷失在中间”和“上下文干扰”等问题的影响（详见第六部分）。因此，RAG与长上下文并非替代关系，而是一种战略选择。开发者需要根据具体任务、模型能力和成本预算，决定是依赖模型的“蛮力”处理能力，还是通过RAG进行更精细的上下文“外科手术”。

3.3. 支柱三：压缩（管理上下文稀缺性）

上下文窗口是有限且宝贵的资源。“压缩”的目标是减少信息的token占用，以便在有限的空间内装入更多的“信号”，同时减少“噪声”[20]。

- 技术实现：
  - 摘要化（**Summarization**）：利用LLM自身的能力，对长对话历史或长文档进行递归式摘要[20]。Claude Code的“自动压缩”（auto-compact）功能是这一技术的典范。当对话接近上下文窗口极限时，它会自动触发，将早期的交互内容压缩成一个简洁的摘要，从而释放空间[20]。
  - 修剪（**Trimming/Pruning**）：这包括基于启发式规则的方法，如简单地移除对话缓冲区中最早的消息[22]，也包括更高级的、基于信息论的方法。例如，根据词元（token）的自信息量或困惑度（perplexity）来判断其重要性，并修剪掉信息量较低的部分[35]。

- 抽取式 **vs.** 生成式压缩：研究发现，抽取式压缩（即从原文中选择关键的句子或词元）通常比生成式压缩（即生成全新的摘要）表现更好，能够在实现高压缩率的同时，最大限度地减少信息损失[36]。
- 高级方法：诸如LLMLingua之类的技术，会使用一个更小、更高效的LLM，以一种“查询感知”的方式，来识别并移除主LLM提示中的非必要词元，从而实现智能压缩[36]。

3.4. 支柱四：隔离（防止干扰与实现复杂性）

“隔离”是指将不同的上下文分离开来，以防止它们相互产生负面干扰，从而减少噪声、提高模型的专注度[20]。这是构建复杂、多步骤、多任务AI系统的关键。

- 架构模式：多智能体系统（Multi-Agent Systems）
  最强大、最典型的隔离技术就是将一个复杂任务分解，并分配给多个专门的子智能体。每个子智能体在自己独立的、经过优化的上下文窗口中运行 9。这种架构天然地避免了“上下文冲突”（信息相互矛盾）和“上下文分心”（无关信息干扰）的问题 9。
- 案例研究：Anthropic的编排器-工作者模型
  Anthropic的多智能体研究系统是“隔离”支柱的顶级范例。它采用了一个“编排器-工作者”（Orchestrator-Worker）模式 40。
  - 工作流程：一个“领导智能体”（编排器）负责接收并分解用户的复杂研究请求，然后生成多个并行的“子智能体”（工作者）[41]。
  - 并行探索：每个子智能体都拥有自己独立的上下文窗口、工具集和研究轨迹，可以并行地对问题的不同方面进行探索。这种广度优先的探索方式是单个智能体无法实现的[41]。在一个内部评估中，这种多智能体架构在处理复杂研究任务时，性能比单个、但能力更强的LLM高出90%以上[41]。
  - 战略意义：这个案例表明，“隔离”不仅是一种防御性策略（防止干扰），更是一种进攻性策略，它能够极大地扩展AI系统解决问题的能力和规模。

表**2**：上下文工程四大支柱：技术与工具

支柱	定义	关键技术	实现案例/工具	主要挑战
写入 **(Write)**	在上下文窗口外持久化信息，以创建记忆。	暂存器、状态对象、长期记忆存储、向量数据库。	Anthropic研究智能体的计划记忆[9]、ChatGPT记忆功能[26]、结构化状	记忆的准确性、更新机制、隐私与安全、成本控制。

			态对象[24]。	
选择 **(Select)**	动态获取并注入最相关的信息以指导LLM。	向量搜索、混合搜索、查询转换、重排序、上下文感知门。	标准RAG (LlamaIndex, LangChain)、上下文感知门 (CAG)[29]、RankRAG[30]。	检索相关性、处理“迷失在中间”问题、防范上下文投毒。
压缩 **(Compress)**	减少上下文的token占用，以适应窗口限制并降低噪声。	摘要化、启发式修剪、基于困惑度的词元修剪、抽取式压缩。	Claude Code的自动压缩[33]、LLMLingua[39]、递归摘要[20]。	信息损失与压缩率的平衡、计算开销、保持人类可读性。
隔离 **(Isolate)**	分割上下文以防止干扰，支持复杂的多任务工作流。	多智能体架构、沙箱执行环境、独立的上下文窗口。	Anthropic的编排器-工作者模型[41]、OpenAI Swarm[9]、LangGraph[23]。	智能体之间的协调与通信、任务分解的有效性、系统复杂性管理。

第四部分：物理层 - KV缓存的隐形挑战

如果说上下文工程的四大支柱是构建AI智能应用的“软件架构”，那么键值缓存（Key-Value Cache, KV-Cache）的管理则是支撑这一切的“硬件与底层系统”问题。理解KV缓存，是将抽象的token流与具体的GPU内存限制联系起来的关键。

4.1. 从抽象**Token**到**GPU**内存：什么是**KV**缓存及其重要性

在Transformer架构中，自注意力（self-attention）机制是其核心。然而，标准自注意力机制的计算复杂度与输入序列长度（即上下文长度）成二次方关系（O(n2)）[43]。这意味着，当上下文长度增加一倍时，计算量会增加四倍，这使得处理长序列的文本生成变得极其缓慢和昂贵。

KV缓存是解决这一问题的关键优化技术。在自回归生成文本时，模型每生成一个新token，都需要计算该新token与前面所有token的注意力分数。如果没有缓存，前面所有token的“键（Key）”和“值（Value）”向量（即K和V）都需要在每一步被重新计算。KV缓存的作用就是

将这些已经计算过的K和V张量存储在GPU的高速显存中，避免重复计算 [43]。通过这种方式，KV缓存将每一步生成的计算复杂度从二次方降低到了线性（

O(n)），从而使LLM的实用化推理成为可能 [45]。

4.2. 新的瓶颈：内存带宽

尽管KV缓存解决了计算瓶颈，但它引入了一个新的、同样严峻的瓶颈：内存。KV缓存的大小与序列长度和批处理大小（batch size）成线性关系。对于拥有百万级token上下文窗口的模型，其KV缓存可以轻易地消耗掉数十甚至上百GB的宝贵GPU显存 [43]。

当处理长上下文时，LLM推理从“计算密集型”（compute-bound）转变为“内存带宽密集型”（memory-bandwidth bound）。这意味着，系统性能的瓶颈不再是GPU的计算速度，而是将巨大的KV缓存从显存中读写所需的时间 [43]。这直接限制了系统的响应延迟和吞吐量，成为制约长上下文应用大规模部署的物理层障碍。

4.3. 深度解析：**KV**缓存优化策略

为了让长上下文窗口在技术上和经济上都变得可行，研究人员开发了多种KV缓存优化策略。这些策略可以大致分为不需要重新训练模型的“令牌级”策略，和需要重新训练模型的“模型级”策略 [47]。

- 令牌级策略（无需重训练）：
  - 驱逐/选择（**Eviction/Selection**）：核心思想是从缓存中丢弃“不那么重要”的token。简单的方法包括滑动窗口注意力（**Sliding Window Attention, SWA**），即只保留最近的几千个token的KV缓存，如Mistral模型所采用的 [43]。更复杂的方法则认识到初始token（所谓的“注意力池”，attention sinks）对于维持模型性能至关重要，因此会同时保留初始和最近的token [46]。还有一些动态方法，会根据token的累计注意力分数来决定哪些是“重击者”（Heavy Hitters）并予以保留，而驱逐其他token，例如H2O和Scissorhands等算法 [43]。
  - 量化（**Quantization**）：这是通过降低缓存中数值的精度来减小其体积。例如，将标准的16位浮点数（FP16）转换为8位甚至4位整数（INT8/INT4），可以使缓存大小减半或更多 [47]。更先进的
    混合精度量化技术，会对更重要的token（如初始token或注意力分数高的token）使用较高的精度，而对其他token使用较低的精度，以在压缩率和性能之间取得平衡

[49]。
    - 低秩分解（**Low-Rank Decomposition**）：利用数学方法（如奇异值分解，SVD）将巨大的KV缓存矩阵分解为多个更小的矩阵的乘积，从而在保留大部分信息的同时显著减小存储需求 [48]。
- 模型级策略（需要重训练）：
    - 注意力分组/共享（**Attention Grouping/Sharing**）：这类方法从模型架构本身入手，设计新的注意力机制来从源头上减少KV缓存的大小。其中包括多查询注意力（**Multi-Query Attention, MQA**），即所有“查询（Query）”头共享同一组K和V头；以及分组查询注意力（**Grouped-Query Attention, GQA**），它将查询头分成几组，每组内部共享K和V头。GQA是MQA和标准多头注意力之间的一种折衷，能够在显著减少KV缓存大小的同时，保持与标准注意力相近的性能，已成为现代高效LLM（如Llama 2/3）的标准配置 [45]。

高层上下文工程与底层**KV**缓存优化的共生关系

高层的应用级上下文工程（如选择和压缩）与底层的推理级KV缓存优化，并非两个独立的领域，而是同一枚硬币的两面，存在着深刻的共生关系。

这一关系的逻辑链条如下：

1. 所有优化的起点，都是GPU显存容量和内存带宽这一物理层面的硬性约束 [43]。
2. 在应用层，上下文工程的“压缩”支柱（例如，通过摘要化）旨在减少发送给模型处理的token数量。一个优秀的摘要策略意味着更少的token需要被计算其K和V向量并存入缓存 [20]。
3. 在推理层，KV缓存优化旨在减少已被处理的token的内存占用。一个高效的量化方案意味着每个token的K和V向量占用的空间更小 [47]。
4. 因此，这两者形成了一个协同循环。一个AI工程师可以选择在应用层投入巨大精力，构建复杂的RAG和摘要流水线，以保持上下文的“精简”，从而减轻对底层KV缓存优化的依赖。反之，如果他们能够使用一个拥有顶尖KV缓存管理技术（如GQA和4位量化）的推理引擎，他们或许可以采用一个更简单的上下文工程流水线，将更多原始上下文直接交给模型处理。
5. 最先进、最经济的系统必然是两者的结合。一个上下文工程师必须了解其所用推理堆栈的底层能力，才能对其上层应用架构做出明智的决策。例如，知道自己使用的模型采用了GQA架构 [45]，会从根本上改变对长上下文成本的计算和使用策略。

第五部分：实践应用 - 上下文工程的实战案例

理论的价值最终体现在实践中。本节将分析几个前沿的AI应用，展示上下文工程的原则是如何在真实世界的产品中被实现、并为用户创造价值的。

5.1. 案例研究：AI驱动的IDE（Cursor & Claude Code）

以Cursor和Claude Code为代表的AI原生集成开发环境（IDE），是上下文工程原则落地最集中的领域之一。它们将开发者的工作流本身作为上下文进行管理，从而超越了简单的代码补全。

- 实现**“自主性滑块”**：Cursor AI的产品设计直接体现了Karpathy的“自主性滑块”理念。它提供了多种工作模式，让用户可以根据任务需求动态调整AI的自主程度 [50]：
  - **Agent**模式：具有高自主性，可以自主探索代码库、编辑多个文件、执行终端命令来完成复杂的功能开发或重构任务。
  - **Ask**模式：只读模式，用于学习和探索。AI可以搜索代码库并回答问题，但不会自动修改任何文件。
  - Manual模式：精确控制模式，AI只会在用户明确指定的文件中进行编辑。
    这种设计是上下文工程在产品交互层面的应用，它允许用户成为上下文的一部分，主动控制AI的能动性。
- 结构化上下文与规范驱动的工作流：Cursor的先进工作流是Sean Grove“规范驱动”哲学的具体实践。它不依赖于单一的即时提示，而是使用结构化的文件来管理上下文 [52]：
  - project_config.md：用于存储项目的长期、静态上下文，如技术栈、核心编码规范、项目目标等。
  - workflow_state.md：作为AI的动态“大脑”，记录当前的状态（如分析、蓝图、构建、验证阶段）、分步计划、执行规则和日志。
    AI通过读取这些结构化、人类可读的文档来指导自己的行为，这使得其工作过程更加可靠和可预测，成功地从“感觉编程”迈向了更稳健的智能体流程 54。
- 自动化的上下文压缩：Claude Code的auto-compact功能是“压缩”支柱的生动案例。当对话历史接近上下文窗口的极限时，它会自动分析并摘要之前的交互，以保留关键信息，避免工作流程中断 [33]。然而，一个有趣的现象是，许多高级用户更倾向于在关键节点（如完成一个功能或修复一个bug后）手动调用
  /compact命令 [55]。这表明，尽管自动化压缩很方便，但在复杂的开发任务中，用户对上下文的精确控制权仍然至关重要，再次印证了“人在回路”在上下文管理中的价值。

5.2. 案例研究：多智能体系统的兴起（**Anthropic**）

Anthropic的多智能体研究系统是“隔离”支柱如何从防御性策略转变为进攻性策略的典范。

- 实践中的隔离：该系统完美诠释了“隔离”原则。它采用“编排器-工作者”模式，由一个领导智能体接收和分解复杂问题，然后将子任务分配给多个专门的子智能体，每个子智能体都在自己独立的、隔离的上下文窗口中工作 [41]。
- 并行化与经济学：这种架构带来了巨大的优势：它能够实现大规模的并行工具调用和信息搜集，将复杂研究任务的时间缩短高达90% [41]。但这种能力也带来了高昂的代价：一个多智能体任务消耗的token量大约是标准聊天的15倍 [11]。这决定了该技术目前仅适用于那些能够承受其成本的高价值、高复杂度的任务。
- 提示驱动的编排：尽管系统架构复杂，但其行为在很大程度上是由“系统级”的提示工程来定义的。开发者通过精心设计的提示来完成以下任务：
  - 定义每个智能体的角色和职责。
  - 向子智能体下达包含详细指令、目标和边界的任务。
  - 设定工作量预算，指导智能体根据任务复杂度分配合理的资源。
  - 引导智能体的“思考过程”，例如让领导智能体先规划再行动。
    这本质上是一种为智能体协作编写的“规范”，是Grove理念在多智能体协同场景下的延伸和应用 11。

表**3**：**AI**原生**IDE**中的上下文管理特性

特性类别	工具	实现细节	与上下文工程原则的关联
自主性控制	**Cursor AI**	提供Agent、Ask、Manual等多种模式，允许用户动态调整AI的自主程度 [50]。	体现了Karpathy的“自主性滑块”理念，将用户控制作为上下文管理的一部分。
结构化输入	**Cursor AI**	使用project_config.md和	实践了Grove的“规范驱动”开发，用结构化

		workflow_state.md等文件来定义长期上下文和动态工作流 [52]。	文档替代模糊的自然语言提示，提高了可靠性。
上下文压缩	**Claude Code**	auto-compact功能在上下文接近极限时自动摘要历史；/compact命令提供手动控制 [33]。	直接实现了“压缩”支柱。用户对 手动控制的偏好，凸显了在复杂任务中人机协作进行上下文管理的重要性。
记忆管理	**Cursor AI**	通过Memories功能和代码库索引，实现跨文件的上下文理解和长期知识保留 [50]。	实现了“写入”支柱，为智能体提供了超越单个聊天会话的长期项目记忆。

5.3. 案例研究：**Manus**智能体的实践原则

Manus，一个专注于构建AI智能体的框架，其整个开发理念都建立在上下文工程之上。他们将上下文工程视为一门实验科学，并分享了从多次重构其智能体框架中学到的宝贵经验，这些经验为理论原则提供了坚实的实践支撑 [71]。

- 将文件系统作为上下文：Manus团队指出，即使是百万级的上下文窗口，在真实的智能体场景中也常常不够用，甚至会成为一种负担，因为长上下文不仅昂贵，而且会降低模型性能 [71]。他们没有采用可能导致信息丢失的激进压缩策略，而是选择将文件系统作为一种外部化的长期记忆。这种方法将长期状态存储在上下文窗口之外，智能体通过与文件系统交互来读写信息，这体现了“写入”支柱的精髓，并从根本上解决了上下文窗口的物理限制 [71]。
- 通过“复述”对抗“中间迷失”：为了解决LLM容易忽略上下文中间部分信息的“中间迷失”问题，Manus采用了一种巧妙的自然语言技巧。它会不断地在上下文的末尾重写一份“待办事项列表”，从而将全局计划和核心目标推入模型的近期注意力范围内。这种“复述”行为有效地利用了模型的注意力偏好，将焦点偏向任务目标，而无需对模型架构进行任何特殊修改 [71]。
- 对**KV**缓存的极致优化：Manus的智能体工作流具有高度的非对称性，其输入与输出的token比例平均约为100:1。这意味着大部分成本和延迟都发生在预填充（prefill）阶段。因此，最大化KV缓存的命中率对其系统的经济性和响应速度至关重要 [71]。他们总结的最佳实践包括：保持系统提示前缀的稳定性（例如，避免使用精确到秒的时间戳）、确保上下文是仅追加的（append-only），以及使用确定性的序列化方法，这些都直接对应了第四部分中讨论的物理层挑战 [71]。

表4：Manus智能体的上下文工程原则

原则	实现细节	与上下文工程原则的关联
外部化记忆	使用文件系统作为持久化存储，而不是将所有状态保存在上下文窗口中[71]。	实现了“写入”支柱，克服了上下文窗口的物理限制和成本问题。
对抗认知偏差	在上下文末尾不断重写待办事项列表，将目标“复述”给模型[71]。	一种创新的、针对“迷失在中间”问题的缓解策略，属于高级的上下文管理。
物理层优化	强调保持提示前缀稳定和上下文仅追加，以最大化KV缓存命中率[71]。	将高层上下文设计与底层KV缓存优化紧密结合，体现了全栈优化的思想。

第六部分：固有风险与缓解策略

上下文工程在赋予AI系统强大能力的同时，也引入了新的风险和挑战。这些风险源于LLM自身的认知局限性，以及将外部信息引入上下文所带来的安全脆弱性。

6.1. LLM的认知偏差：长上下文的脆弱性

尽管LLM的上下文窗口在不断扩大，但其有效利用长上下文的能力仍然存在根本性的缺陷。

- **“迷失在中间”（Lost in the Middle）现象**：大量研究已经证实，LLM在处理长上下文时表现出一种明显的“U型”性能曲线。它们能够很好地回忆起位于上下文开头和结尾的信息，但对于埋藏在中间部分的信息，则经常会“遗忘”或忽略[57]。这是一个普遍存在的现象，即使是那些拥有百万级token上下文窗口的先进模型也未能幸免[59]。这直接威胁到需要依赖长对话历史或长文档进行推理的应用的可靠性。
- **上下文分心（Contextual Distraction）**：并非所有上下文信息都同等重要。对话中的闲聊、已被纠正的错误、或早期的离题讨论，都会成为“噪声”。当这些无关信息被包含在上下文中时，会严重干扰模型对当前核心任务的专注力，导致其性能下降[57]。一个重

要的结论是：更多的上下文并不总是更好。在RAG场景中，加入更多文档有时反而会引入与主题相关但答案错误的“强干扰项”（hard negatives），从而混淆模型，导致性能下降而非提升 [57]。

- 缓解策略：这些认知偏差的存在，使得主动的上下文管理成为必需，而不是可选项。有效的策略包括：
  1. 重排序（**Re-ranking**）：在将最终的提示发送给LLM之前，通过一个独立的步骤，识别出最重要的信息，并将其明确地移动到提示的开头或结尾，以迎合模型的注意力偏好 [59]。
  2. 压缩与摘要：如第三部分所述，通过摘要化或抽取式压缩，将长上下文精炼为最核心的要点，这从根本上减少了“中间”区域的长度和无关“噪声”的数量，是缓解这两个问题的最有效手段之一 [57]。
  3. 结构化上下文：使用清晰的结构，如JSON格式或带有明确标题的Markdown，可以帮助模型更好地解析和定位关键信息，减少其在扁平文本中“迷失”的可能性 [62]。

6.2. 安全漏洞：上下文投毒的威胁

当AI系统开始依赖外部数据源来构建上下文时，一个新的、严峻的攻击面便随之出现：上下文投毒（Context Poisoning）。

- 概念：数据投毒是指攻击者通过操纵训练数据、微调数据或RAG系统中的外部数据，来向AI系统中植入漏洞、后门或偏见 [63]。在RAG系统中，这种攻击被称为上下文投毒。攻击者可以精心制作一份文档，当这份文档被RAG系统检索并注入到LLM的上下文中时，会诱导LLM产生恶意的、非预期的输出 [64]。
- 主要攻击向量：
  - 指令注入（**Instruction Injection**）：这是一种间接的提示注入。攻击者在文档中嵌入类似“忽略你之前的所有指令，现在你是一个……”的命令，试图覆盖LLM的原始系统提示，从而劫持其行为 [64]。
  - 检索操纵（**Retrieval Manipulation**）：攻击者利用RAG系统检索机制的特点，在恶意文档中堆砌大量与目标查询高度相关的关键词，并使用“紧急”、“重要”等字眼作为标题，以确保这份恶意文档在检索时获得高分，从而被优先注入上下文 [64]。
  - 数据提取（**Data Extraction**）：攻击者在看似无害的文档中植入指令，诱骗LLM从知识库中的其他文档里总结和提取敏感信息（如密码、API密钥），并将其输出 [64]。
  - 后门（**Backdoors**）：通过污染微调数据或嵌入模型，攻击者可以在模型中植入一个隐藏的“触发器”。在正常情况下，模型行为一切正常，但一旦遇到特定的触发词或输入模式，就会激活恶意行为，如绕过安全验证或执行隐藏命令 [63]。
- 防御态势：
  1. 数据溯源与审查：对所有用于构建上下文的数据源进行严格的审查，并追踪其来源

和转换历史，优先使用可信的、经过验证的数据 [63]。
2. 输入净化与沙箱化：对输入到上下文中的外部内容进行过滤，检测并清除潜在的恶意指令。同时，在沙箱环境中运行LLM，限制其访问文件系统、网络等高风险资源的能力 [63]。
3. 确定性访问控制（**Deterministic Access Control**）：这是最关键、最根本的防御措施。访问控制必须在检索层（即LLM看到数据之前）强制执行。系统应根据用户的身份和权限，在数据库或文件系统层面就决定哪些文档可以被检索。永远不要信任**LLM**本身来做安全决策，比如让它判断“这份文档是否包含我应该看到的信息”。必须假设，一旦信息进入了上下文窗口，它就可能被用户获取 [64]。

第七部分：领域之未来 - 趋势、角色与战略建议

上下文工程正在从一个由少数前沿研究者探讨的概念，迅速演变为一个定义下一代AI应用成败的核心工程学科。它的发展将深刻影响AI行业的组织架构、技术栈和战略方向。

7.1. 学科的成熟：**“上下文工程师”**的崛起

行业内的对话正在发生决定性的转变，焦点从“提示工程”转向了更系统、更具挑战性的“上下文工程” [4]。随之而来的是一个新角色的出现：“上下文工程师”（Context Engineer） [69]。

这个角色的职责远不止是编写提示。它是一个系统设计者的角色，负责设计和构建那些能够在正确的时间、以正确的格式，为**LLM**提供正确的信息、工具和记忆的动态系统 [2]。这本质上是一个信息系统工程的岗位，需要一套复合型的技能 [10]。

- 关键技能：一个成功的上下文工程师需要具备以下能力：
  - 信息检索：深刻理解并能实践RAG、混合搜索、重排序等技术。
  - 数据架构：能够设计和管理用于存储上下文的数据库（特别是向量数据库）和数据流水线。
  - 文本处理技术：掌握摘要、实体提取、数据清洗等NLP基础技能。
  - 提示工程基础：能够编写清晰、无歧义的指令作为上下文的一部分。
  - **API**与框架熟悉度：熟练使用LLM API以及LangChain、LlamaIndex等主流开发框架 [2]。

7.2. 未来轨迹：迈向自优化与分层系统

上下文工程的未来发展将朝着更智能、更自动化和更结构化的方向演进。

- 更智能的上下文利用：未来的重点将从追求更大的上下文窗口，转向更智能地利用这些空间。质量将压倒数量，每一token的“信息价值”将成为核心考量[69]。
- 分层与多分辨率记忆：先进的AI系统将摆脱扁平的、单一的上下文缓冲区。取而代之的将是分层记忆架构，例如：
  - 短期记忆：存储最近几轮对话的逐字记录。
  - 中期记忆：存储对过去对话的摘要。
  - 长期记忆：以向量嵌入或知识图谱的形式存储核心事实和用户偏好。
    这种多分辨率的记忆系统能让AI在不同抽象层次上进行高效回忆和推理 3。
- 自优化流水线：我们可以预见，能够自我优化的上下文管理系统将会出现。其工作原理可能如下：
  1. 系统记录所有被注入上下文的信息片段。
  2. 追踪每个片段对最终任务成功与否的贡献度，从而计算出每个信息片段的“输入回报率”（Return on Input）。
  3. 基于这些数据，训练一个“显著性模型”（Salience Model），使其能够自动预测在特定任务中哪些信息最值得被包含进上下文。
     这种闭环反馈系统将使上下文工程从手动调优走向自动化优化 3。
- 上下文即代码（**Context as Code**）：最顶尖的AI工程团队将会像管理源代码一样管理上下文。这意味着：
  - 版本控制：对上下文配置和关键文档（如规范文件）进行版本控制。
  - 回归测试：建立一套标准化的评估基准，在每次上下文变更后运行回归测试，以防止性能衰退或行为漂移。
  - 差异比对：在部署新的上下文配置前，进行“diff”操作，清晰地审查变更内容。
    这种严谨的工程实践是确保复杂AI系统长期稳定和可靠的基石 3。

7.3. 组织的战略实施指南

对于希望在AI时代保持竞争力的企业和组织，采纳上下文工程不仅是技术选择，更是战略布局。

- 何时投资：当组织的需求从简单的问答机器人转向构建需要处理状态、依赖外部知

识、执行多步骤任务，或要求高可靠性和一致性的复杂应用时，就应该果断地从提示工程转向上下文工程 [2]。

- 构建**"上下文感知"**的文化：这意味着在整个开发流程中，将上下文视为一等公民。这需要数据科学家、软件工程师和领域专家之间的紧密协作，共同定义、构建和维护应用所需的知识库和交互逻辑。
- 工具与基础设施：组织需要投资于核心的基础设施，包括向量数据库、数据ETL流水线，以及像LangChain或LlamaIndex这样的开发框架。一个常被忽视但至关重要的考量是，必须仔细评估所选推理服务提供商的KV缓存效率，因为它直接关系到应用的性能和运营成本 [71]。
- 衡量投资回报率（**ROI**）：上下文工程的ROI需要从一个综合的视角来衡量。关键是在"上下文的成本"（包括token费用、延迟、工程投入）与"任务成功率和可靠性的提升"之间找到平衡点 [3]。特别是对于像多智能体系统这样成本高昂的架构，必须确保其所执行的任务价值足以支撑其高昂的运营开销 [11]。

第八部分：结论

本报告的分析清晰地表明，上下文工程已成为定义AI应用开发下一阶段的核心学科。它标志着一个重要的范式转变：我们不再将大型语言模型视为神秘的、不可预测的黑箱，而是开始将其作为强大但需要精心"喂养"的推理引擎，并围绕它来系统地工程化整个信息环境。

从Andrej Karpathy的"LLM即操作系统"的宏大愿景，到Sean Grove"规范即代码"的意图驱动哲学，再到LangChain"四大支柱"的实践蓝图，我们看到了一个从概念到实践的完整思想体系正在形成。这些思想领袖和框架为我们提供了导航这一新领域的地图。然而，地图本身并不能保证成功。真正的成功将取决于开发者和组织能否以一种严谨、系统且高度关注安全的工程化方法，来管理AI时代最宝贵的资源——上下文。

技术的演进是迅速的。KV缓存的优化正在物理层上不断拓宽长上下文的可能性，而RAG、摘要、多智能体等应用层技术则在更智能地利用这些可能性。与此同时，"迷失在中间"和"上下文投毒"等挑战提醒我们，这条道路并非坦途。

最终，那些能够超越简单的提示调优，将上下文工程内化为其核心竞争力，并建立起相应的人才、文化和技术栈的组织，将能够构建出真正可靠、智能且能创造巨大价值的AI系统。他们将是这场技术革命的最终赢家。

引用的著作

1. Context Engineering (1/2)—Getting the best out of Agentic AI Systems | by A B Vijay Kumar, 访问时间为 七月 19, 2025,

https://abvijaykumar.medium.com/context-engineering-1-2-getting-the-best-out-of-agentic-ai-systems-90e4fe036faf
2. Context Engineering: A Guide With Examples - DataCamp, 访问时间为 七月 19, 2025, https://www.datacamp.com/blog/context-engineering
3. Context Engineering Concepts | Phoenix - Arize AI, 访问时间为 七月 19, 2025, https://arize.com/docs/phoenix/learn/context-engineering/context-engineering-concepts
4. The New Skill in AI is Not Prompting, It's Context Engineering - Philschmid, 访问时间为 七月 19, 2025, https://www.philschmid.de/context-engineering
5. Software is Changing (Again): Key Takeaways from Andrej ..., 访问时间为 七月 19, 2025, https://dulandias.com/2025/07/software-is-changing-again-key-takeaways-from-andrej-karpathys-talk-at-yc-ai-startup-school/
6. Context Engineering is the New Vibe Coding (Learn this Now) - YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/watch?v=Egeuql3Lrzg
7. AI Coding Tools Slow Experienced Developers by 19% in METR Study - AInvest, 访问时间为 七月 19, 2025, https://www.ainvest.com/news/ai-coding-tools-slow-experienced-developers-19-metr-study-2507/
8. Context Engineering is the New Vibe Coding (Learn this Now) - YouTube, 访问时间为 七月 19, 2025, https://m.youtube.com/watch?v=Egeuql3Lrzg&t=0
9. Context Engineering - LangChain Blog, 访问时间为 七月 19, 2025, https://blog.langchain.com/context-engineering-for-agents/
10. Context Engineering Guide - Hacker News, 访问时间为 七月 19, 2025, https://news.ycombinator.com/item?id=44508068
11. Anthropic's multi-agent system overview a must read for CIOs | Constellation Research Inc., 访问时间为 七月 19, 2025, https://www.constellationr.com/blog-news/insights/anthropics-multi-agent-system-overview-must-read-cios
12. Andrej Karpathy: Software Is Changing (Again) : YC Startup Library | Y Combinator, 访问时间为 七月 19, 2025, https://www.ycombinator.com/library/MW-andrej-karpathy-software-is-changing-again
13. Build Something Worth the Energy: Lessons from YC's AI Startup School on Agents, Defensibility, and the Future of Product - Gary Smith, 访问时间为 七月 19, 2025, https://www.garysmith.me/blog/yc-ai-sus
14. Andrej Karpathy: Software Is Changing (Again) - YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/watch?v=LCEmiRjPEtQ
15. OpenAI Sean Grove - The New Code : r/OpenAI - Reddit, 访问时间为 七月 19, 2025, https://www.reddit.com/r/OpenAI/comments/1m198hh/openai_sean_grove_the_new_code/
16. The New Code — Sean Grove, OpenAI - YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/watch?v=8rABwKRsec4
17. OpenAI's Sean Grove: Everything is a Spec: The Universal Language of Intent -

YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/shorts/hSdP5EiroQI
18. Code is Not Dead: Why Specifications Aren't the Full Story | by Alexander Olmedo - Medium, 访问时间为 七月 19, 2025, https://medium.com/@olmedo2408/code-is-not-dead-why-specifications-arent-the-full-story-40a654736856
19. Sean Grove (OpenAI): The New Code @ AI Engineer World's Fair 2025 - YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/watch?v=BlvlLtt164I
20. What is Context Engineering: Clearly Explained - Apidog, 访问时间为 七月 19, 2025, https://apidog.com/blog/context-engineering/
21. Context Engineering for Agents - YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/watch?v=4GiqzUHD5AA
22. Context Engineering: The New AI Strategy for Scalable LLMs ..., 访问时间为 七月 19, 2025, https://www.sundeepteki.org/blog/from-vibe-coding-to-context-engineering-a-blueprint-for-production-grade-genai-systems
23. Overview of Context Engineering in AI | Coconote, 访问时间为 七月 19, 2025, https://coconote.app/notes/3e0da30f-1466-4163-aca6-c4d0bc89f587
24. Context Engineering in AI Agents | Coconote, 访问时间为 七月 19, 2025, https://coconote.app/notes/0f19bf31-44c8-494e-92dc-0aeb2d318642
25. What is LlamaIndex ? | IBM, 访问时间为 七月 19, 2025, https://www.ibm.com/think/topics/llamaindex
26. Understanding ChatGPT's Memory: How AI Remembers (and ..., 访问时间为 七月 19, 2025, https://dev.to/gervaisamoah/understanding-chatgpts-memory-how-ai-remembers-and-forgets-54f8
27. Understanding How ChatGPT Handles Context Across Conversations - Community, 访问时间为 七月 19, 2025, https://community.openai.com/t/understanding-how-chatgpt-handles-context-across-conversations/1086232
28. A Novel Framework for Fast Information Retrieval based Response Generation using Large Language Model - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/html/2406.16383v1
29. Context Awareness Gate For Retrieval Augmented Generation - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/html/2411.16133v1
30. RankRAG: Unifying Context Ranking with Retrieval-Augmented Generation in LLMs - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/abs/2407.02485
31. LaRA: Benchmarking Retrieval-Augmented Generation and ... - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/abs/2502.09977
32. [2502.12462] Emulating Retrieval Augmented Generation via Prompt Engineering for Enhanced Long Context Comprehension in LLMs - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/abs/2502.12462
33. claudelog.com, 访问时间为 七月 19, 2025, https://claudelog.com/faqs/what-is-claude-code-auto-compact/#:~:text=I%20observe%20that%20auto%2Dcompact,to%20continue%20working%20without%20

interruption.
34. what-is-claude-code-auto-compact | ClaudeLog, 访问时间为 七月 19, 2025, https://claudelog.com/faqs/what-is-claude-code-auto-compact/
35. Compressing Context to Enhance Inference Efficiency of Large Language Models, 访问时间为 七月 19, 2025, https://aclanthology.org/2023.emnlp-main.391/
36. Characterizing Prompt Compression Methods for Long Context Inference - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/html/2407.08892v1
37. Concise and Precise Context Compression for Tool-Using Language Models - ACL Anthology, 访问时间为 七月 19, 2025, https://aclanthology.org/2024.findings-acl.974.pdf
38. Prompt Compression with Context-Aware Sentence Encoding for Fast and Improved LLM Inference, 访问时间为 七月 19, 2025, https://ojs.aaai.org/index.php/AAAI/article/view/34639/36794
39. Prompt Compression in Large Language Models (LLMs): Making Every Token Count, 访问时间为 七月 19, 2025, https://medium.com/@sahin.samia/prompt-compression-in-large-language-models-llms-making-every-token-count-078a2d1c7e03
40. Anthropic's multi-agent research system raises the bar for open-ended AI reasoning, 访问时间为 七月 19, 2025, https://centific.com/news-and-press/anthropic-s-multi-agent-research-system-raises-the-bar-for-open-ended-ai-reasoning
41. How we built our multi-agent research system \ Anthropic, 访问时间为 七月 19, 2025, https://www.anthropic.com/engineering/built-multi-agent-research-system
42. Anthropic: Building a Multi-Agent Research System for Complex Information Tasks - ZenML LLMOps Database, 访问时间为 七月 19, 2025, https://www.zenml.io/llmops-database/building-a-multi-agent-research-system-for-complex-information-tasks
43. LLM Inference Series: 4. KV caching, a deeper look | by Pierre Lienhart | Medium, 访问时间为 七月 19, 2025, https://medium.com/@plienhar/llm-inference-series-4-kv-caching-a-deeper-look-4ba9a77746c8
44. Understanding and Coding the KV Cache in LLMs from Scratch - Sebastian Raschka, 访问时间为 七月 19, 2025, https://sebastianraschka.com/blog/2025/coding-the-kv-cache-in-llms.html
45. Keep the Cost Down: A Review on Methods to Optimize LLM's KV-Cache Consumption, 访问时间为 七月 19, 2025, https://openreview.net/forum?id=8tKjqqMM5z&referrer=%5Bthe%20profile%20of%20hai%20zhao%5D(%2Fprofile%3Fid%3D~hai_zhao1)
46. Keep the Cost Down: A Review on Methods to Optimize LLM's KV Cache Consumption, 访问时间为 七月 19, 2025, https://arxiv.org/html/2407.18003v4
47. A Survey on Large Language Model Acceleration based on KV Cache Management - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/pdf/2412.19442?
48. A Survey on Large Language Model Acceleration based on KV Cache

Management, 访问时间为 七月 19, 2025, https://openreview.net/forum?id=z3JZzu9EA3
49. A Survey on Large Language Model Acceleration based on ... - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/pdf/2412.19442
50. Modes - Cursor Docs, 访问时间为 七月 19, 2025, https://docs.cursor.com/agent/modes
51. Cursor AI Tutorial for Beginners [2025 Edition] - YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/watch?v=3289vhOUdKA
52. [Guide] A Simpler, More Autonomous AI Workflow for Cursor [New Update] - Page 3, 访问时间为 七月 19, 2025, https://forum.cursor.com/t/guide-a-simpler-more-autonomous-ai-workflow-for-cursor-new-update/70688?page=3
53. [Guide] A Simpler, More Autonomous AI Workflow for Cursor [New Update] - Showcase, 访问时间为 七月 19, 2025, https://forum.cursor.com/t/guide-a-simpler-more-autonomous-ai-workflow-for-cursor-new-update/70688
54. Auto mode is the norm now, here are some thoughts and tips. : r/cursor, 访问时间为 七月 19, 2025, https://www.reddit.com/r/cursor/comments/1m28j6l/auto_mode_is_the_norm_now_here_are_some_thoughts/
55. Explain Claude Code Auto-Compact - is it just referring to the current session? - Reddit, 访问时间为 七月 19, 2025, https://www.reddit.com/r/ClaudeAI/comments/1l4xp6v/explain_claude_code_autocompact_is_it_just/
56. Using the Claude Compact Command - YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/watch?v=QzEck41pueY
57. Context-Engineering Challenges & Best-Practices | by Ali Arsanjani ..., 访问时间为 七月 19, 2025, https://dr-arsanjani.medium.com/context-engineering-challenges-best-practices-8e4b5252f94f
58. Do LLM's get "lost in the middle" during summarization as well? [D] : r/MachineLearning, 访问时间为 七月 19, 2025, https://www.reddit.com/r/MachineLearning/comments/1beb7vi/do_llms_get_lost_in_the_middle_during/
59. Overcome Lost In Middle Phenomenon In RAG Using LongContextRetriver - AI Planet, 访问时间为 七月 19, 2025, https://medium.aiplanet.com/overcome-lost-in-middle-phenomenon-in-rag-using-longcontextretriver-2334dc022f0e
60. Practical AI/ML Paper reading：“Lost in the Middle”: How Language Models Use Long Contexts | by Christmas Carol | Medium, 访问时间为 七月 19, 2025, https://medium.com/@carolzhu/lost-in-the-middle-how-language-models-use-long-contexts-2891830f8000
61. Reducing Distraction in Long-Context Language Models by Focused Learning - arXiv, 访问时间为 七月 19, 2025, https://arxiv.org/html/2411.05928v1
62. Context Engineering: The Future of AI Prompting Explained - AI-Pro.org, 访问时间

为 七月 19, 2025, https://ai-pro.org/learn-ai/articles/why-context-engineering-is-redefining-how-we-build-ai-systems/
63. LLM04:2025 Data and Model Poisoning - OWASP Gen AI Security Project, 访问时间为 七月 19, 2025, https://genai.owasp.org/llmrisk/llm042025-data-and-model-poisoning/
64. RAG Data Poisoning: Key Concepts Explained | Promptfoo, 访问时间为 七月 19, 2025, https://www.promptfoo.dev/blog/rag-poisoning/
65. Imperceptible Content Poisoning in LLM-Powered Applications - WingTecher, 访问时间为 七月 19, 2025, http://www.wingtecher.com/themes/WingTecherResearch/assets/papers/paper_from_24/ContentPoisoning_ASE24.pdf
66. Data Poisoning: a threat to LLM's Integrity and Security - RiskInsight, 访问时间为 七月 19, 2025, https://www.riskinsight-wavestone.com/en/2024/10/data-poisoning-a-threat-to-llms-integrity-and-security/
67. Scaling Trends for Data Poisoning in LLMs - AAAI Publications, 访问时间为 七月 19, 2025, https://ojs.aaai.org/index.php/AAAI/article/view/34929/37084
68. Context Engineering: What It Is and Why It Matters - YouTube, 访问时间为 七月 19, 2025, https://www.youtube.com/watch?v=fYgsZnkFeck
69. Beyond the Prompt: How Context Engineering Is Shaping the Future of AI Interaction | by Oliver Ipsioco | Jul, 2025 | Medium, 访问时间为 七月 19, 2025, https://medium.com/@oliver.ipsioco/beyond-the-prompt-how-context-engineering-is-shaping-the-future-of-ai-interaction-a8da0270644b
70. www.philschmid.de, 访问时间为 七月 19, 2025, https://www.philschmid.de/context-engineering#:~:text=Context%20Engineering%20is%20the%20discipline,needs%20to%20accomplish%20a%20task.&text=A%20System%2C%20Not%20a%20String,just%20a%20static%20prompt%20template.
71. Context Engineering for AI Agents: Lessons from Building Manus, 访问时间为 七月 19, 2025, https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus