贵州轻工职业技术学院

2026 届毕业设计

# FlowCabal
# Agent 半自动化写作辅助工具


学科专业：大数据技术

指导老师：曾小爽

学生学号：20230105235

学生姓名：田照涛


中国．贵州．贵阳

2026 年 3 月

# 摘　要

FlowCabal 是一款专注于 AI 辅助写作的软件，专门面向高质量长篇写作的场景。软件提供蓝图式工作流编排、内置记忆文件和 agent 交互方式。本文将从产品功能、技术选型、软件架构、交互设计和具体用例等方面介绍这款软件。软件源代码在 https://github.com/isirin1131/FlowCabal 开源

**关键词**：工作流；智能体；记忆

## Abstract

FlowCabal is a software focused on AI-assisted writing, specifically designed for high-quality long-form writing scenarios. The software provides blueprint-style workflow orchestration, built-in memory files, and agent interaction methods. This article introduces the software from aspects such as product features, technology selection, software architecture, interaction design, and specific use cases. The source code of the software is open-sourced at https://github.com/isirin1131/FlowCabal.



**Keywords**: workflow; agent; memory

# 目　录

# 第 1 章 绪论

## 1.1 研究背景

2017 年，Vaswani 等人提出的 Transformer 架构[1] 为大语言模型（LLM）的发展奠定了基础。此后短短数年间，LLM 经历了爆发式增长：2020 年 Brown 等人发布的 GPT-3[2] 以 1750 亿参数和少样本学习能力震动了整个领域，2022 年底 ChatGPT 的上线将基于 LLM 的对话式 AI 带入大众视野，2023 年的 GPT-4[3] 更是将上下文窗口从千级 tokens 推向数万级。到 2026 年初，主流模型的上下文窗口已普遍达到百万 token 级别。在这一进程中，AI 辅助写作始终是最受关注的应用场景之一。

当基于 LLM 的 chat-ai-app 刚刚进入人们视野时，第一个被广泛关注的概念是**提示词工程**。GPT-3[2] 所展示的少样本提示能力表明，精心设计的提示词可以在无需微调的前提下引导模型完成各类任务；Wei 等人提出的思维链提示[4] 则进一步证明，在提示中加入中间推理步骤可以显著提升模型在复杂推理任务上的表现；White 等人在[5] 中进一步将提示词设计归纳为可复用的模式目录，标志着提示词工程从经验总结走向了方法论。然而提示词工程的关注点集中于单次调用的输入设计，当任务从简单问答扩展到需要多步协作、持续记忆和外部工具调用的复杂场景时，仅靠优化提示词已然不够。

2025 年年中，Andrej Karpathy 在一篇广泛传播的帖子[6] 中提出了**上下文工程**这一概念，指出在工业级 LLM 应用中，真正的挑战不是写一条好的提示词，而是“用恰当的信息填充整个上下文窗口的精细艺术与科学”。上下文工程自然地涵盖了提示词工程，但视野更广：不仅关注“对模型说什么”，还关注在什么时机、以什么结构、用多少信息来构建上下文。这一概念在 2025 年到 2026 年初的 AI 编程大爆发和 agent 大爆发中得到了充分的实践验证——Anthropic 和 OpenAI 的 agent 团队都在上下文管理与记忆方面积累了大量公开经验（详见研究意义一节）。

在 AI 辅助写作这个具体领域，提示词工程和上下文工程同样是最核心的挑战。Yang 等人的 DOC[7] 通过详细大纲控制提升了长篇故事的连贯性，证明了结构化控制手段对长文本生成的重要性——但这仍然停留在单次模型调用的层面。当写作规模扩大到数万字乃至更长的篇幅时，单次调用的方案无论如何优化都会撞上上下文窗口的硬性天花板。如何在多次调用间维持一致性、如何管理跨章节的记忆、如何编排复杂的写作流程——这些问题已经超出了提示词工程的范畴，呼唤更系统性的上下文工程方案。

## 1.2 研究意义

尽管 LLM 的上下文窗口已经从 2022 年的 4096 tokens 扩展到了百万级别，但 Wu 等人在 LongGenBench[8] 中的评测表明，十个主流模型在长文本生成任务中的表现均随文本长度增加而显著下降——模型能“读”长文本，不代表能“写”长文本。与此同时，Hong 等人提出的“上下文腐化”现象[9] 指出，输入 token 数量的增加会系统性地削

弱模型的推理能力，这对需要在数万字篇幅内维持连贯性的写作场景构成了根本性的挑战。

事实上，工业界最前沿的 agent 团队已经在上下文管理和记忆方面积累了大量经验。Anthropic 在 2024 年 12 月发表的《Building Effective Agents》[10] 中将增强型 LLM（配备检索、工具和记忆的 LLM）定义为 agent 系统的基本构建单元，并明确区分了“工作流”和“agent”两种编排范式。随后 Young 在《Effective Harnesses for Long-Running Agents》[11] 中进一步指出，长时间运行的 agent 的核心挑战在于每个新的会话都从零开始、没有前序记忆，他们的解决方案是通过初始化 agent 配合编码 agent 的双 agent 架构，辅以进度文件来在会话间传递状态。2026 年初，Anthropic 又在 Claude 开发者平台上正式推出了上下文编辑和记忆工具[12]，前者能在逼近 token 上限时自动清除陈旧的工具调用结果（在 100 轮 web 搜索评测中减少了 84% 的 token 消耗），后者则通过基于文件的系统让 agent 在上下文窗口之外存取信息。

OpenAI 的 Codex 团队也在同期给出了类似的答案。Bolin 在《Unrolling the Codex Agent Loop》[13] 中详述了 Codex CLI 的 agent 循环机制，其中上下文压缩（compaction）是关键策略——当 token 数超过阈值时，系统会用摘要替换原始对话，使 agent 能在有限的窗口内持续工作。Choi 在《Run Long Horizon Tasks with Codex》[14] 中则展示了一个实际案例：GPT-5.3-Codex 在不间断运行约 25 小时、消耗约 1300 万 token 的条件下生成了约 3 万行代码，其中最关键的技术是持久化项目记忆——将规格、计划、约束和状态写入 markdown 文件供 agent 反复回访，以防止长时间运行中的漂移。

另一方面，工作流编排已经在 agent 领域展现出了强大的组织能力。Fan 等人的 WorkflowLLM[15] 在 ICLR 2025 上证明了 LLM 可以通过工作流来编排复杂的多步骤任务，LangGraph 等框架也已将节点式工作流变成了 agent 应用的经典实践。然而，这些工作几乎都聚焦于代码生成、数据处理等结构化任务，将工作流编排应用于创意写作的尝试仍然稀缺。

FlowCabal 的意义就在于填补这个空白：上述团队在编程 agent 上验证过的上下文压缩、持久化记忆、进度文件等机制，完全有潜力迁移到长篇写作的场景中。FlowCabal 将工作流编排、记忆管理和 agent 监控三者结合，把一个原本高度依赖人类逐句干预的长篇写作过程变成半自动化的流水线，同时也为上下文工程在创作领域的落地提供了一个可供参考的实践。

# 第 2 章 产品功能

本节以渐进式的视角审视了 FlowCabal 的功能设计，省略了很多软件实现上的细节，力求以更通用的视角解释清楚 FlowCabal 的优点。

## 2.1 蓝图式工作流编排

研究背景一节已经指出，长篇写作必然涉及对 LLM 的多次调用。由此而来的核心问题是：如何编排这些调用之间的依赖关系，以及如何将前序调用的输出转化为后续调用的输入？蓝图式工作流编排是 FlowCabal 对这一问题的回答。

将 LLM 抽象为 `Model(query) = Answer`，常见的 chat-ai-app 以线性对话的形式组织 `query`：

```
"query": [
    { "role": "user", "content": "你好" },
    { "role": "assistant", "content": "你好！有什么我可以帮助你的吗？" },
    { "role": "user", "content": "你是谁" },
    ......
]
```

这种线性结构意味着每次调用都携带完整的对话历史，既浪费 token，也无法灵活定义调用间的依赖关系。节点工作流提供了更通用的替代方案：每个 LLM 调用被封装为独立节点，节点间通过显式连线声明数据依赖。Unreal Engine 的蓝图系统、ComfyUI 以及 agent 领域的 LangGraph 都采用了这一范式。

FlowCabal 的节点工作流可以形式化描述如下。设工作流包含 $n$ 个节点 $M_1, M_2, ..., M_n$，第 $k$ 个节点的查询由自定义的组装函数 $f_k$ 从其依赖节点的输出中生成：

$$M_k.\text{query} = f_k(\{m.\text{Answer} \mid m \in D_k\})$$

其中 $D_k \subset \{M_1, ..., M_n\}$ 是 $M_k$ 的依赖集。例如，一个简单的拼接函数 $f(a, b, c) = a + b + c$ 即可将三个依赖节点的输出合并为一条查询。这些节点间的依赖关系构成有向无环图（DAG），FlowCabal 使用 Kahn 算法进行拓扑排序后逐层并行执行。

值得注意的是，这种节点式调用天然具备上下文压缩的特性：每个节点 $M_k$ 的查询仅包含其依赖节点的输出，而非整个工作流的完整历史——$M_k$ 的查询不会出现在任何非依赖节点 $M_p$ 的查询中。这恰好呼应了研究背景中的上下文工程问题：节点工作流通过结构化的依赖声明，从机制层面实现了对上下文的精确控制。

## 2.2 内置记忆文件

长篇写作中，Agent 需要在整部小说——包括已定稿的章节、角色设定、世界观和文体约定——中探索上下文并检查一致性。常见的做法是引入 RAG（检索增强生成），用向量数据库进行语义召回。然而 FlowCabal 明确放弃了这一路径：小说中语义相似

的段落可能多达数十上百种，embedding 召回无法区分哪个是当前真正需要的；更关键的是，伏笔与回收在语义空间里往往距离很远，余弦相似度捕捉不到因果链。

FlowCabal 采用纯文件系统的记忆架构，核心思路是将记忆文件视为定稿章节（manuscripts）的**有损缓存**。类比编程 agent：代码库是记忆，grep 是检索；对应到 FlowCabal：`manuscripts/` 目录存放完整的定稿章节作为信息源，而 `characters/`、`world/`、`voice.md` 等记忆文件则是为了避免每次都加载全部原文而提取的结构化摘要。

记忆的加载遵循三级渐进策略：L0 层是 `index.md` 导航索引，Agent 每次启动时自动加载；L1 层是各类记忆文件，Agent 读取 L0 后通过工具按需加载；L2 层是 `manuscripts/` 中的完整原文，通过文件间的稀疏跳转链接按需可达。这种设计使得上下文检索是**因果关系驱动**的——Agent 理解叙事上下文后自主决定加载什么，而非依赖统计意义上的语义相似性。所有记忆文件对人和 Agent 完全开放：用户可以直接用文本编辑器修改，Agent 也可以在写作过程中创建新文件或更新已有内容。

## 2.3 Agent 交互方式

FlowCabal 的 Agent 不仅负责文本生成，还承担上下文注入和约束检查的职责。其核心机制是 **agent-inject**：节点的 prompt 中可以嵌入注入点（hint），执行时 Agent 会以节点 prompt 和上游输出为锚点，从记忆系统中查询相关约束，并将查到的上下文自动注入。这实质上将约束查询和上下文注入合二为一——Agent 为了检查一致性而查到的信息，恰好就是生成时所需的上下文。

工作流的执行采用增量构建模式。每个节点的 prompt 经解析后计算 SHA-256 哈希值，与缓存中的哈希比对：匹配则跳过，不匹配则重新执行并级联更新所有下游节点。这意味着用户修改一个节点的 prompt 后再次运行，只有受影响的节点会重新生成，大幅节省 token 消耗和等待时间。

每个节点的输出以多版本形式保留，版本切换仅修改当前指针而不删除旧版本。用户既可以在不同版本间对比择优，也可以直接手动编辑某个节点的输出作为新版本。此外，工作流支持 step 模式——逐层执行，每层完成后暂停等待用户确认。用户可在暂停期间审视结果、调整参数或编辑输出，然后再推进到下一层。这种逐层暂停的机制将一个高度自动化的流水线转变为人机协作的迭代过程。

# 第 3 章 技术选型

## 3.1 前端技术

## 3.2 后端技术

## 3.3 AI 模型接入

# 第 4 章 软件架构

## 4.1 总体架构

## 4.2 核心模块设计

## 4.3 数据流设计

# 第 5 章 交互设计

## 5.1 界面布局

## 5.2 用户操作流程

# 第 6 章 具体用例

## 6.1 用例一

## 6.2 用例二

# 第 7 章 总结与展望

# 参考文献


[1] VASWANI A, SHAZEER N, PARMAR N, 等. Attention Is All You Need[C/OL]//Advances in Neural Information Processing Systems: 卷 30. 2017. https://arxiv.org/abs/1706.03762.

[2] BROWN T B, MANN B, RYDER N, 等. Language Models are Few-Shot Learners[C/OL]//Advances in Neural Information Processing Systems: 卷 33. 2020: 1877-1901. https://arxiv.org/abs/2005.14165.

[3] OPENAI. GPT-4 Technical Report[EB/OL]. (2023). https://arxiv.org/abs/2303.08774.

[4] WEI J, WANG X, SCHUURMANS D, 等. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models[C/OL]//Advances in Neural Information Processing Systems: 卷 35. 2022. https://arxiv.org/abs/2201.11903.

[5] WHITE J, FU Q, HAYS S, 等. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT[EB/OL]. (2023). https://arxiv.org/abs/2302.11382.

[6] KARPATHY A. Context Engineering[EB/OL]. (2025). https://x.com/karpathy/status/1937902205765607626.

[7] YANG K, KLEIN D, PENG N, 等. DOC: Improving Long Story Coherence With Detailed Outline Control[C/OL]//Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL). 2023. https://arxiv.org/abs/2212.10077.

[8] WU Y, HEE M S, HU Z, 等. LongGenBench: Benchmarking Long-Form Generation in Long Context LLMs[C/OL]//Proceedings of the International Conference on Learning Representations (ICLR). 2025. https://arxiv.org/abs/2409.02076.

[9] HONG K, TROYNIKOV A, HUBER J. Context Rot: How Increasing Input Tokens Impacts LLM Performance[R/OL]. (2025-07). https://research.trychroma.com/context-rot.

[10] SCHLUNTZ E, ZHANG B. Building Effective Agents[EB/OL]. (2024). https://www.anthropic.com/research/building-effective-agents.

[11] YOUNG J. Effective Harnesses for Long-Running Agents[EB/OL]. (2025). https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents.

[12] ANTHROPIC. Managing Context on the Claude Developer Platform[EB/OL]. (2026). https://www.anthropic.com/news/context-management.

[13] BOLIN M. Unrolling the Codex Agent Loop[EB/OL]. (2026). https://openai.com/index/unrolling-the-codex-agent-loop/.

[14] CHOI D. Run Long Horizon Tasks with Codex[EB/OL]. (2026). https://developers.openai.com/blog/run-long-horizon-tasks-with-codex.

[15] FAN S, CONG X, FU Y, 等. WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models[C/OL]//Proceedings of the International Conference on Learning Representations (ICLR). 2025. https://arxiv.org/abs/2411.05451.

# 致　谢

感谢我的家人、老师和朋友。

# 原 创 性 声 明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导教师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究在做出重要贡献的个人和集体，均已在文中以明确方式标明。本人在导师指导下所完成的毕业设计（论文）及相关作品，知识产权归属贵州轻工职业技术学院。本人完全意识到本声明的法律责任由本人承担。

作者签名：_____　　日期：_____年____月____日

# 关于毕业设计（论文）使用授权的声明

本人完全了解贵州轻工职业技术学院有关保留、使用毕业设计（论文）的规定，同意学校保留或向国家有关部门或机构送交毕业设计（论文）的复印件和电子版，允许毕业设计（论文）被查阅和借阅；本人授权贵州轻工职业技术学院可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存毕业设计（论文）和汇编本论文。

本毕业设计（论文）属于：

保　密（　　），在_____年解密后适用授权。

不保密（　　）

(请在以上相应方框内打“√”)

作者签名：_____　　导师签名：_____

日　期：_____年____月____日