

Context Rot & Attention Budget

上下文腐烂与注意力预算——窗口装得下，不代表模型读得懂

2026年10月19日 | AI Agent 知识分享 | 第80期

一、一句话概括

上下文腐烂 (Context Rot) 是指随着上下文窗口里的 Token 数量增长，模型准确召回其中信息的能力反而下降的现象。它不是窗口太小的问题，而是注意力预算 (Attention Budget) 被摊薄的问题：模型对上下文的利用率并不均匀，开头和结尾的信息记得住，埋在中间的关键信息会被「吃掉」。

对测试工作的直接含义：验证 Agent 不能只问「答案对不对」，必须同时验证「信息放在什么位置」「注入多少轮之后还能不能召回」。同样的 Prompt，把关键事实从开头挪到第 40 轮，召回率可能掉一半——这类缺陷靠常规的功能回归测试是测不出来的。

二、核心原理

>> 1. 注意力预算是有限资源

模型在生成时让每个 Token 去关注其他 Token，代价随 Token 数量呈二次方增长。上下文越长，单条信息分到的注意力越少，召回准确率随之下降。Anthropic 在 2025 年 9 月的工程博客中把这件事命名为 context rot，并给出一句很硬的话：必须把上下文当作一个收益递减的有限资源来管理。

>> 2. 与窗口大小无关

这是最容易误判的一点：窗口从 20 万扩到 100 万，并不代表模型能更可靠地使用它。实际观测到的现象是，即使标称支持百万级 Token，模型在 5 万 Token 上下就开始明显退化。扩大窗口只是推迟退化，而不是阻止退化。

>> 3. Lost in the Middle 的 U 形曲线

Liu 等人在 TACL 2024 发表的论文里给出了经典结论：模型在上下文的最开头和最结尾召回最好，关键事实被埋在中间时性能下降超过 30 个百分点，形成一条 U 形曲线。

>> 4. 四个直接成因

位置效应：关键事实埋在中段；注意力稀释：注入内容越多，单条信息权重越低；指令冲突：旧指令与新事实在窗口里互相干扰；轮次衰减：多轮对话中前面轮次的约束力逐轮衰减。

>> 5. 实测数据

Chroma 的 Context Rot 研究 (2025 年 7 月) 测了 18 个主流模型，包括 GPT-4.1、Claude 4、Gemini 2.5、Qwen3，结论是模型并未均匀使用上下文，输入越长表现越不可靠，且退化幅度在不同模型间差异很大、常常反直觉。

>> 6. 压缩策略不是免费的

Anthropic 给出三条长任务路径：服务端压缩（推荐，自动做且 Token 计量更准）、结构化笔记（把关键决策显式写回上下文）、子智能体（每个子任务在独立窗口里完成，只把 1000 到 2000 Token 的结论回传）。关键在于：压缩会丢信息，丢了什么必须能测出来。

核心要点

上下文腐烂与窗口大小无关：窗口扩到 100 万，退化只是被推迟，不是被阻止

Lost in the Middle：头尾记得住，中间被吃掉，中段召回可掉 30 个百分点以上

测 Agent 要同时测三件事：答案对不对、信息放在哪个位置、第几轮之后还能召回

生产预算上限取实测拐点的 60% 到 70%，不要直接采用官方标称窗口

数值型字段保真率必须 100%：金融场景不接受任何一位小数的巧合

三、测试实战

测试一：位置效应扫描（Lost in the Middle 复现）

- 用同一个模型、同一套问题，准备 21 个候选事实（fact_01 到 fact_21），只有 1 个是正确答案。
- 依次把正确答案分别放在位置 1、3、5、7、9、11、13、15、17、19、21，其余位置填充无关但语义相近的内容。
- 每个位置跑 3 次取多数结果，共 33 次调用。
- 工具：自己写 30 行 Python 循环即可，不需要专门框架；要可视化就用 RAGAS 的 Faithfulness 指标做交叉验证。
- 评估指标：各位置召回准确率、U 形曲线是否存在、头尾位置与中段位置的准确率差值。
- 判定线：中段比头尾低 10 个百分点以上，说明该模型的位置效应显著，生产环境必须固定关键信息的落位。

测试二：长度梯度衰减测试

- 固定正确答案位置在开头，把总长度从 1k、5k、10k、20k、50k、100k、200k Token 逐档拉长，填充内容保持同类语义。
- 工具：脚本自动生成填充语料，用 tiktoken 精确计数。
- 评估指标：召回准确率随长度的衰减曲线、P95 响应时延、每档 Token 成本。
- 判定线：找出准确率开始陡降的拐点 Token 数，这就是该模型在生产里的实际上限，比官方标称值更有用。

测试三：工具结果保真度测试

- Agent 先调用工具拿到一段 JSON 结果（含账号、金额、日期等字段），把结果注入上下文。
- 接着插入 3 轮无关对话，再让 Agent 基于该结果做后续动作。
- 工具：一个返回固定 JSON 的 Mock Server 即可。
- 评估指标：字段保真率（每个字段单独统计）、数值抄错率、格式破坏率。
- 判定线：数值型字段保真率必须 100%，任何一位小数抄错都算失败，因为金融场景不接受四舍五入的巧合。

测试四：历史指令保持率测试

1. 在第 1 轮注入约束，例如「输出必须是 JSON」「金额单位是元」「禁止推测」。
2. 分别在第 3、5、10、20 轮插入新指令并观察 Agent 是否仍然遵守旧约束。
3. 工具：自己写脚本做规则匹配校验。
4. 评估指标：约束违反率随轮次的变化曲线、首次违反发生在第几轮。
5. 判定线：核心安全约束（越权、涉密、格式破坏）的违反率必须为 0，出现一次就要调整上下文装配顺序或改用结构化笔记。

测试五：子智能体上下文隔离测试

1. 主智能体只收结论摘要（1000 到 2000 Token），子智能体在独立窗口里处理全量资料。
2. 对照实验：同一批任务分别走单智能体全量上下文和子智能体隔离上下文。
3. 工具：任何支持多轮会话的模型 API 都可以搭，对照组用 Anthropic 公开报告的多智能体研究系统作为参照。
4. 评估指标：任务完成率对比、Token 消耗对比、跨窗口信息丢失率。
5. 判定线：隔离方案在完成率不低于对照组的前提下，Token 消耗应显著下降，否则说明摘要压缩丢掉了关键信息。

测试六：压缩触发点回归测试

1. 开启自动压缩（例如 Anthropic Agent SDK 的 `compaction_control`，默认阈值 100000 Token）。
2. 构造超长任务，观察压缩是否触发、触发时保留了几条关键决策。
3. 评估指标：压缩触发次数、压缩前后关键决策保留率、压缩后任务完成率变化。
4. 判定线：压缩后完成率下降超过 5 个百分点，说明摘要策略有信息损失，需要自定义 `summary_prompt` 或在压缩前把关键决策写进结构化笔记。

补充建议：这 6 项测试全部沉淀成 CSV 用例集（用例编号、前置条件、注入位置、预期结果、实测结果、判定线），就能接 JMeter 或 Python Runner 做每日回归。

四、常见坑点

坑点一：把窗口上限当成实际可用容量

官方标称 20 万或 100 万 Token，实测可能在 5 万就开始退化。正确做法是先跑长度梯度测试找出拐点，再用拐点的 60% 到 70% 作为生产预算上限。

坑点二：关键信息塞在中间位置

长文档检索结果、历史对话摘要、工具返回的大量 JSON 经常被拼在上下文中段，正好落在 U 形曲线的谷底。对策：目标事实放开头（系统提示区），最近结论放结尾（紧贴提问），中段只放低优先级参考材料。

坑点三：压缩后不回归就上线

压缩会丢信息，而且丢的往往是你当时觉得不重要的那条。压缩策略改一次就要跑一轮完成率回归，重点看跨轮次的长程任务，短任务看不出来。

坑点四：拿短任务数据集评估长上下文

用几十条一问一答的样本评估 Agent，测不出腐烂。必须构造长度梯度和轮次梯度两类样本，覆盖 1 到 20 轮的会话深度。

坑点五：只测召回不测时延和成本

上下文越长，P95 时延和单次成本同步上涨。验收报告里必须同时给出准确率、P95 时延、每任务 Token 成本三个数，缺一个都会得出错误结论。

五、延伸阅读

- Anthropic 工程博客 «Effective context engineering for AI agents» : <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>
- Lost in the Middle 原始论文 (Liu et al., TACL 2024) : <https://arxiv.org/abs/2307.03172>
- Chroma «Context Rot: How Increasing Input Tokens Impacts LLM Performance» : <https://research.trychroma.com/context-rot>
- Anthropic «Effective harnesses for long-running agents» : <https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents>
- Claude 上下文编辑与压缩文档: <https://platform.claude.com/docs/en/build-with-claude/context-editing>
- RAGAS 评估框架 (Faithfulness 等指标) : <https://docs.ragas.io>
- tiktoken 分词计数工具: <https://github.com/openai/tiktoken>