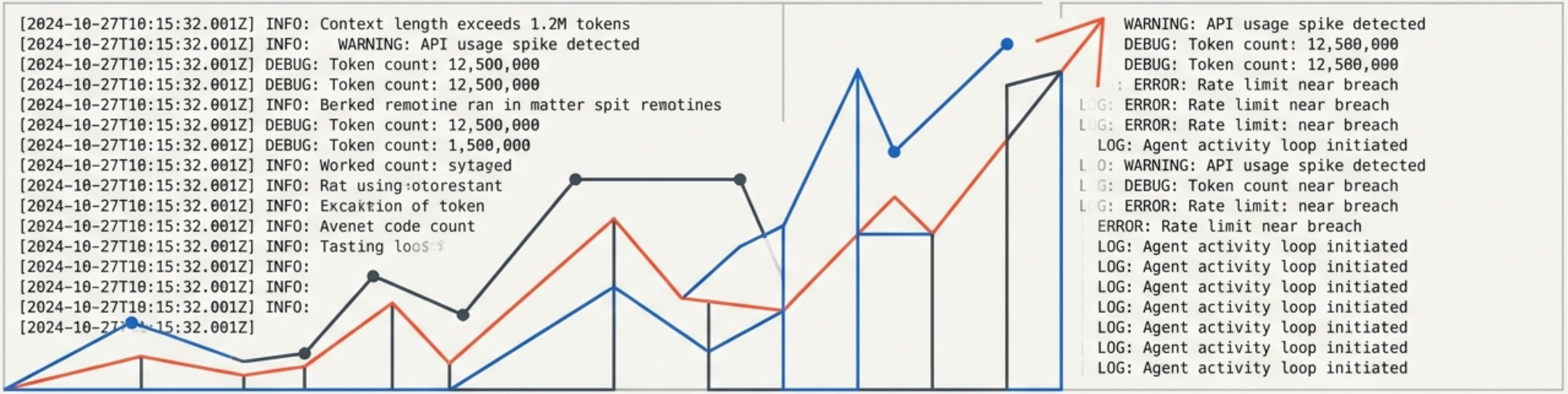


Token 账单取证：千万级上下文的赛博悬案

一次由 Claude Code 驱动的 Agent 成本解剖与长上下文生命周期重构

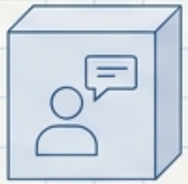
TARGET_SYSTEM:	OpenClaw Framework (AI 伴侣)
MODEL:	Gemini Pro (1M Context)
INCIDENT:	Extreme API Cost Overrun
INVESTIGATOR:	Claude Code & System Logs

离谱的账单倍率：当正常闲聊触发了指数级 API 调用

用户真实发送消息数（2.5 天）

20

对象：具备情绪与记忆的 AI 伴侣



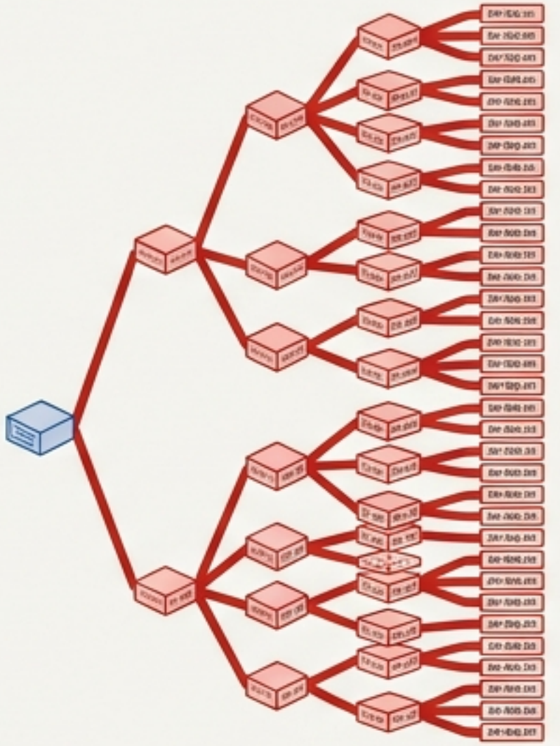
DANGEROUS DISCREPANCY DETECTED

> TIME ELAPSED: 2.5 DAYS

系统调用对话轮次

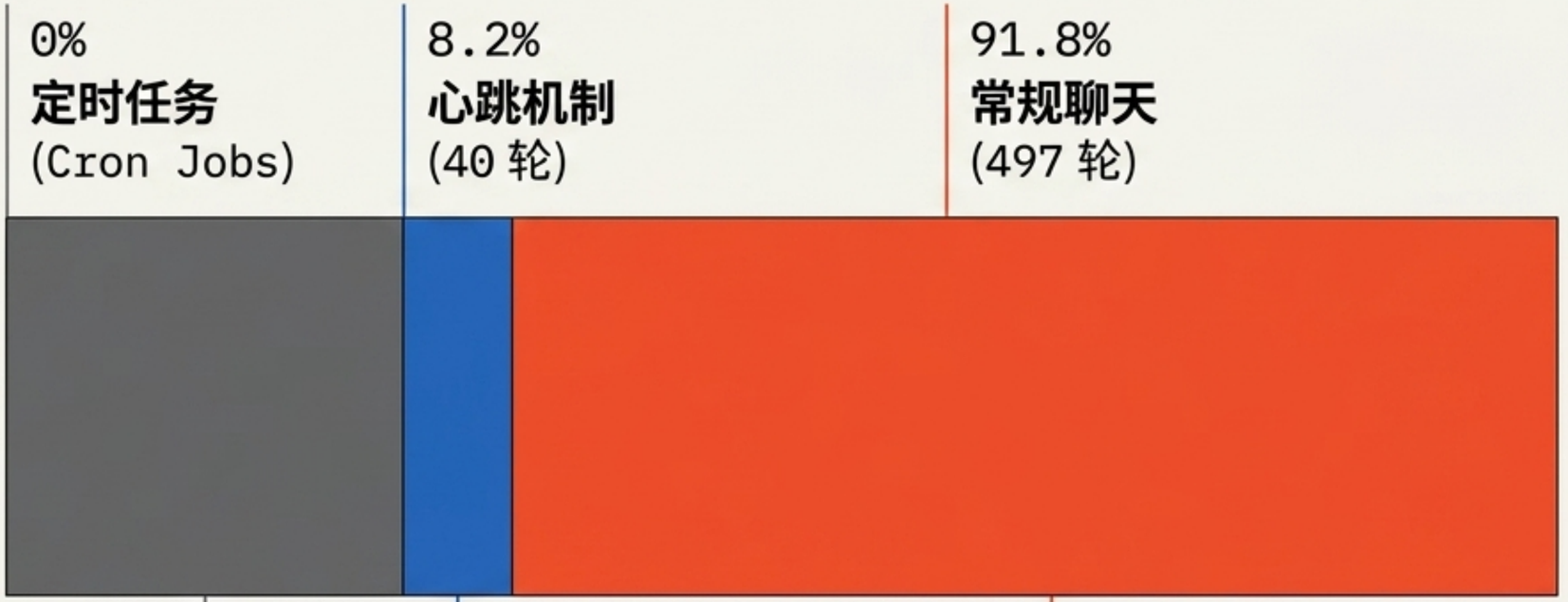
537

历史极值（2月17日）：
约 30 条用户消息引发 750 轮系统调用



[SYSTEM_WARNING] 每一轮系统调用，都是一次完整的 API 请求，并重新发送不断膨胀的全部上下文。我们面临的不是简单的聊天机器人，而是一台失控的 Token 焚烧炉。

第一轮审讯：谁在后台疯狂制造轮次？

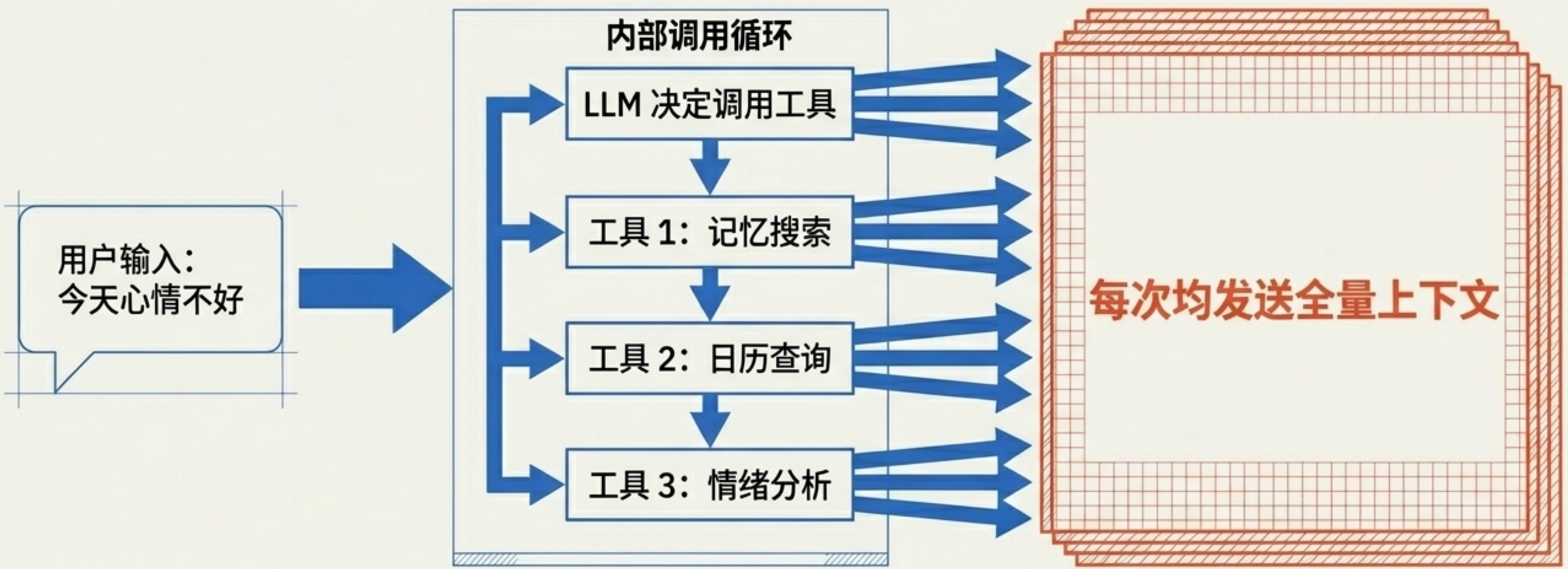


分析结论：

问题并未出在后台的复杂自动化任务上，而是完全集中在最基础的**常规聊天链路中**。

每一轮闲聊都在比上一轮更加昂贵。

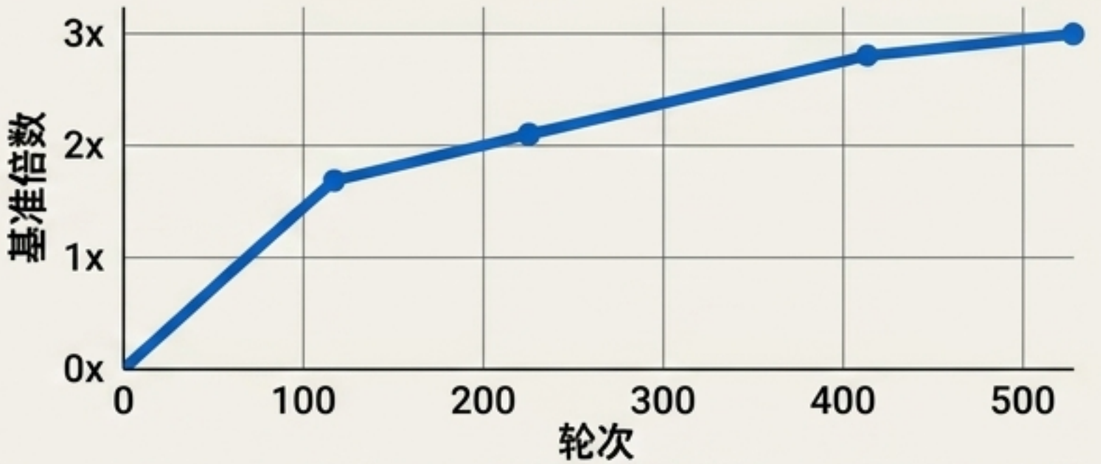
乘数效应：30 句闲聊如何裂变成 750 轮 API 调用？



框架鼓励 Agent 使用工具，却没有限制其对上下文的放大效应。一条消息触发四轮全量调用，导致交互比例从 1:1 畸变为惊人的 25:1。

强行撬开黑盒：单调递增的系统内存

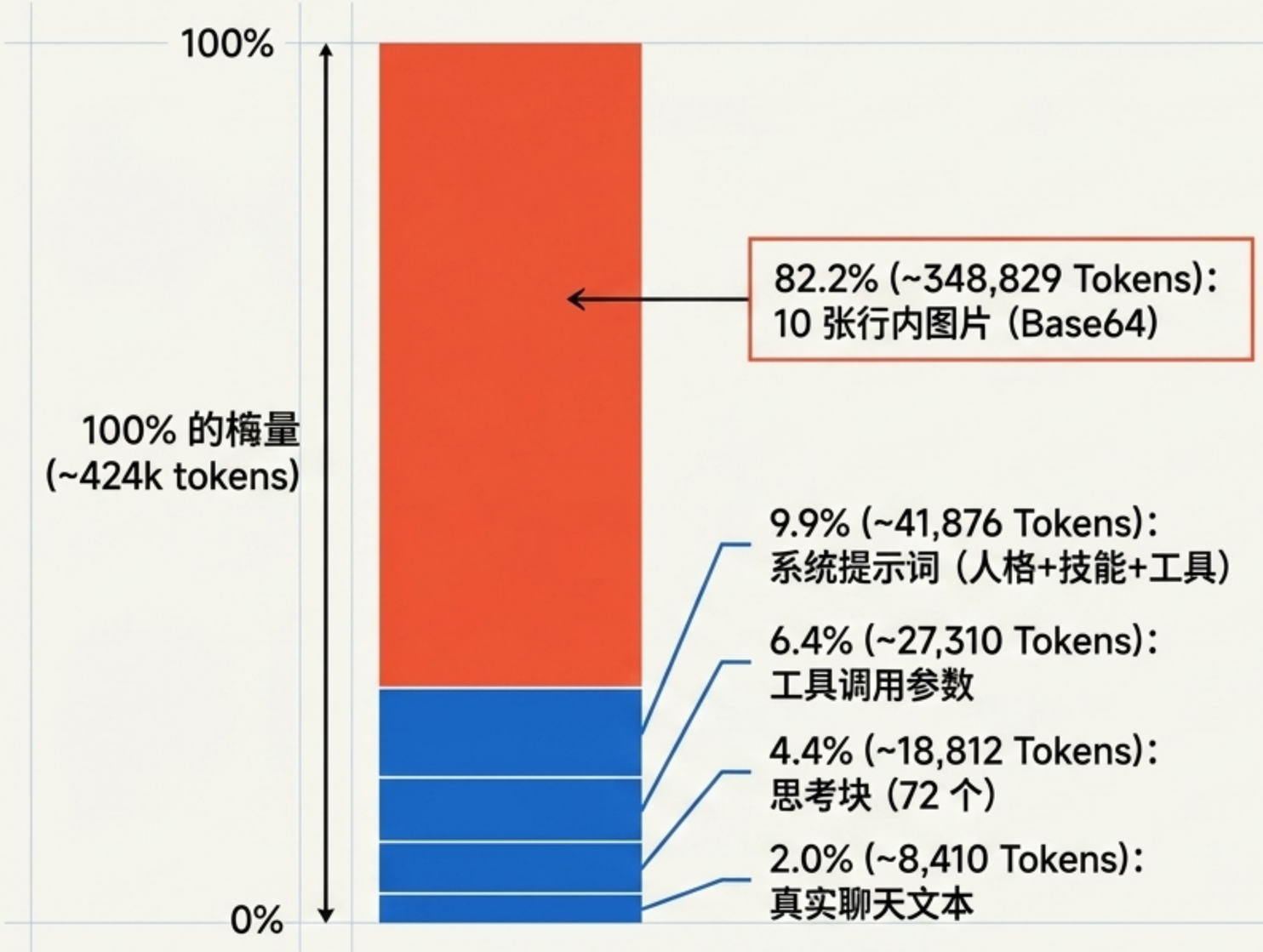
```
> Claude Code's Probe Scripts

$ run_diagnostics.sh

[Attempt 1] -> SyntaxError (Python f-string)
- 错误信息: string formatting error, line 45.

[Attempt 2] -> FileNotFoundError
- 错误信息: log file 'system_audit.log' not
found.

[Attempt 3] -> KeyError (数据 schema 不一致)
- 错误信息: key 'session_id' missing from JSON
object.

[Attempt 4] -> 成功拉取 JSONL 日志数据
- 数据流建立成功，开始分析。
```



轮次	时间戳	内存使用（基准）
0	02-25 14:02	基准水位
105	02-26 00:50	1.7x 基准
210	02-26 18:51	2.1x 基准
420	02-27 10:43	2.8x 基准
536	02-27 23:44	3.0x 基准

结论：没有下降，没有平台期。上下文只涨不缩，呈现致命的单调递增。

致命解剖：第 390 轮的上下文里到底装了什么？



缺乏多模态垃圾回收机制，导致两天半内积累的 10 张图片永远不被清除。

每一次基础的 API 调用（甚至只为发一句早安），都在无意义地反复重发这 35 万 Token 的历史盲区。

法医诊断矩阵：被高估的表象与幽灵般的死代码

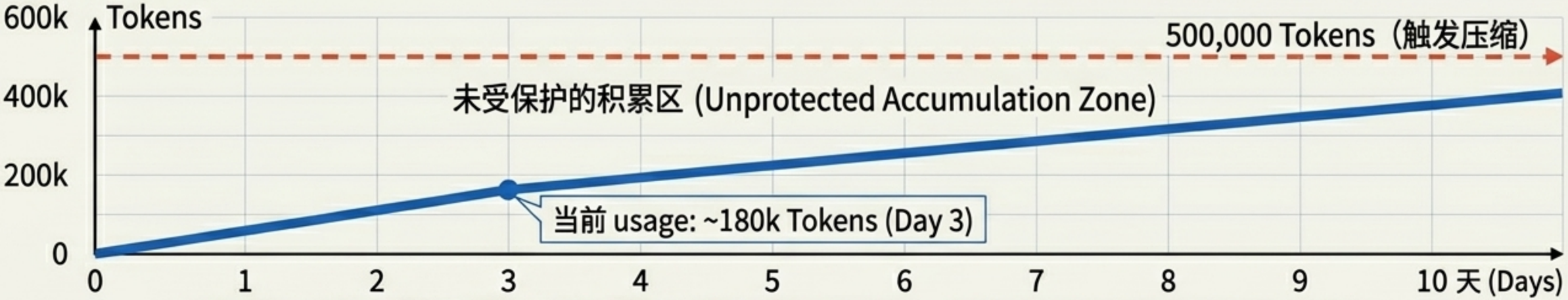
直觉与假设（预期的耗钱与省钱点）	代码真相（真实的运行状态）
假设 1（耗钱元凶）： 日志中存在巨大的 72 个思考块与 257 次工具调用记录，必必然在大量消耗上下文。	**真相 1（已有的防线）：** 配置 `dmStripToolHistory: true` 早就生效。思考块仅存在于本地日志中，并未发送给模型。直觉严重高估了其实际开销。
假设 2（省钱屏障）： 已开启 `cache-ttl` 修剪模式，旧内容会自动转化为廉价缓存。	**真相 2（隐形的漏洞）：** 该修剪代码被硬编码为仅对 Anthropic 模型生效。当前使用的 Gemini 模型完全跳过了该逻辑，处于没有任何防线的裸奔状态。**死代码**在安静地漏钱。

百万上下文的陷阱：537 轮零次压缩的灾难组合

公式：灾难的数学逻辑

`Gemini Pro 1M 物理上限` × `compaction.mode: safeguard (逼近极限才触发)` × `ratio: 0.5` = **50 万 Token 的压缩触发死线**

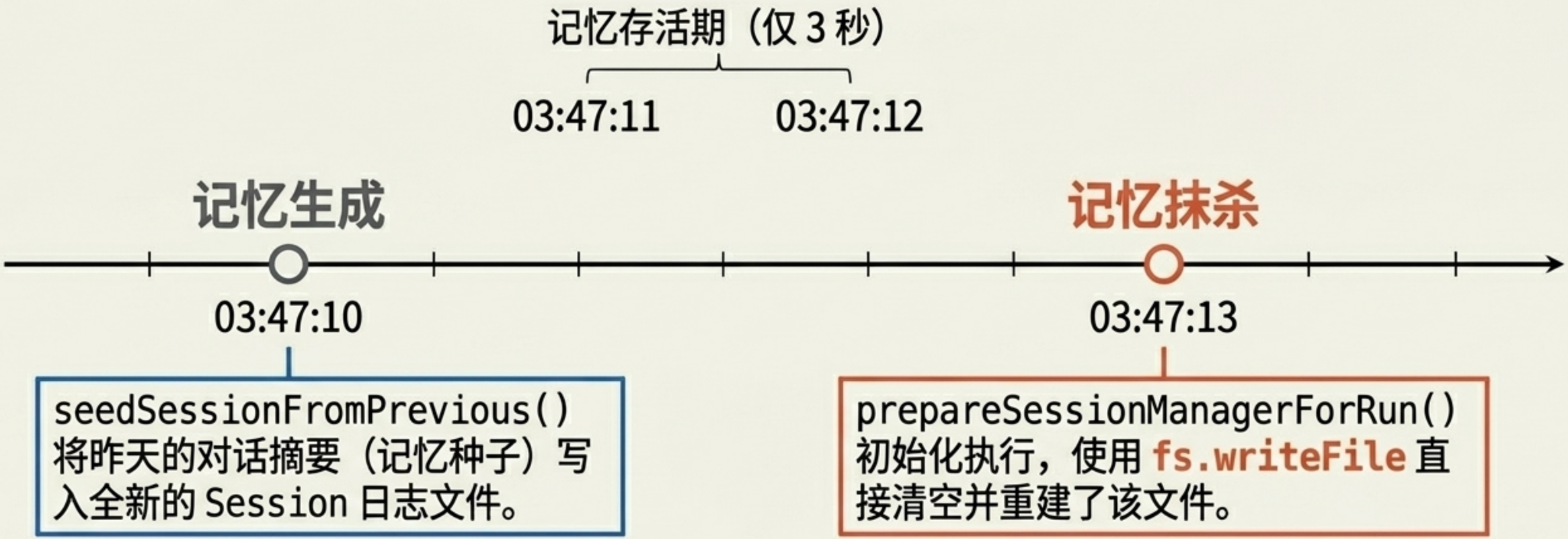
图表：危险的阈值与漫长的等待



图示表明，实际使用曲线需要数天甚至数周才能触及安全网，期间账单在无防护状态下持续增长。

> 1M 上下文窗口常被视为大模型的营销卖点。但对于缺乏生命周期管理的伴侣 Agent 而言，它意味着你的账单可以连续滚雪球数天，而不会触发任何自动清理。主动管理上下文不是可选项，而是必需品。

幽灵 Bug：每天早晨失忆的赛博魅魔



> 长线记忆仅存活了 3 秒钟。因为没有任何代码崩溃或报错日志，AI 只是每天早上对昨天的互动显得有些茫然，导致这个致命的系统漏洞得以长期潜伏。

封堵系统性漏水点：双管齐下的干预方案

配置项干预 – Config Patch

- `session.reset.mode: daily`
  -> 强制每日切断，防止图片跨天无限制积累
- `agents.defaults.contextTokens: 200,000`
  -> 下调物理上限，将压缩触发点强行降至 10 万 Token
- `thinkingDefault: low`
  -> 伴侣闲聊降级推理深度，减少输出消耗
- `heartbeat.every: 2h + historyLimit: 20`
  -> 降低心跳频次，并为心跳实施严苛的 20 轮短记忆隔离

代码层重构 – Code Refactoring

- **复活死代码：**新增 `contextPruning.mode: always`，绕过模型限制，强行激活修剪机制。
- **多模态垃圾回收（GC）：**突破系统保护，当图片滑出最近消息区，将其降维替换为纯文本占位符 `[Image removed from context]`。
- **记忆注入改造：**放弃直接写文件，改将记忆种子作为新 Session 首条消息的前缀挂载，避免被重置覆盖。

全局洞察：面向生产环境的 Agent Token 经济学

一、主动的生命周期切割

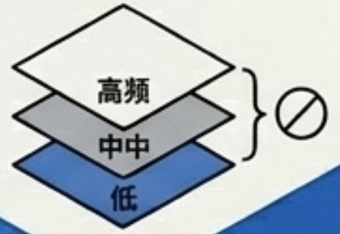
永远不要迷信无限上下文。必须强制干预 Session 生命周（如强制日结），大模型的 Context Window 只是计算缓存，绝非长期数据库。

二、多模态资源的独立 GC

图片/文件绝对不能与纯文本同权管理。重型资产必须具备独立的衰减机制，否则单张图片将永远拖累每一轮 API 调用的成本底盘。

三、分层计算成本模型

剥离高频低智任务的上下文堆叠。心跳探测等动作必须硬隔离历史记录，或降级使用极简廉价模型，拒绝让打个招呼支付高昂的 Token 过路费。



CASE CLOSED

“**成本调试是一场严密的取证工作**。死代码是隐形的，真正的答案只存在于 Token 级的显微镜下。现有的实验框架证明了 AI 伴侣在产品侧走得通，但这笔高昂的账单证明了——**如果你不把 Token 经济学作为系统的一等公民，你在商业上永远走不起**。”

[NEXT_ARCHIVE] 引出下一阶段：重构地基 ——《为什么我决定从零开始造 Mio Framework》