跨会话的永久黑匣子

当 pi-experiencev2 遇上 billion-context

刘政东 · 2026-03-24

工程师的两个极端困境

- **极端 A：单会话死撑到底**
  - 不敢关终端，Prompt 堆到 150K Token
  - 模型注意力稀释、胡言乱语
  - 首字延迟拉长到 30 秒，账单爆炸
- **极端 B：重开窗口瞬间失忆**
  - 新开干净 Session，Agent 变“盲人”
  - 丢失历史架构决策与接口约定

常见方案的两大死穴

常见 Agent 跨会话“长记忆”方案的两大工程死穴

死撑不退导致崩溃，盲目注入打碎物理缓存

死穴 A：死撑不退的“超级单 Session”

不敢关窗口、不敢开新会话，试图在一个上下文里跑一辈子

第 1 天任务：环境配置、依赖安装 (30K Token)

第 3 天任务：重构数据层、修报错 (60K Token)

第 7 天任务：写文档、改接口 (80K Token)

后果 1：注意力严重稀释

塞满几百次命令输出，早期的核心规矩被模型遗忘

后果 2：首字延迟高达 30 秒，账单爆炸

每次随手回一句，都要重新 Prefill 几十万 Token

死穴 B：粗暴的“向量 RAG 动态注入”

开新会话，每次把检索出的 5 条碎片历史硬塞进 Prompt

每次发送消息时的动态 Prompt 结构：

动态注入的 5 条历史片段 (每次请求内容与字数都不同!)

系统指令与用户当前任务输入...

后果 1：彻底打碎 Prompt Caching

头部内容每轮都在变，前缀哈希永远失配，缓存命中率 0%

后果 2：上下文污染与信息断章取义

向量切片丢失了完整的调用链与状态机，模型经常误判

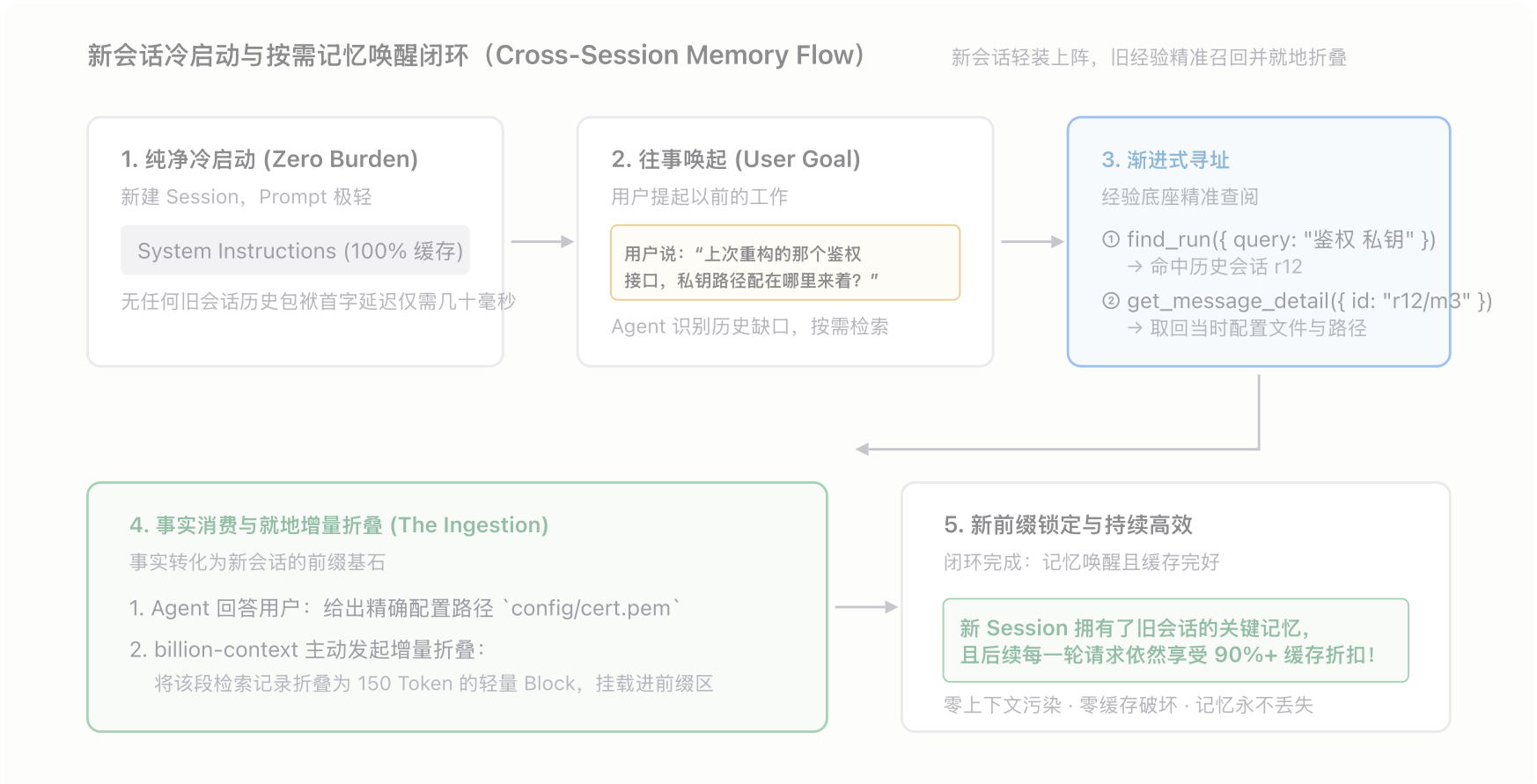
pi-experiencev2：物理磁盘黑匣子

- **RUN2 原生架构**：基于轻量 SQLite WAL，会话轨迹毫秒级落盘
  - `runs` 表：记录会话元数据与百字导航摘要
  - `messages` 表：完整记录全部推理、工具参数与返回值
- **双轨检索机制**：
  - `scope: "summary"`：极低 Token 扫目录锁定会话 ID
  - `scope: "content"`：穿透到全量记录做精准匹配
- **工程安全红线**：
  - 自动屏蔽超大 Base64，>12KB 自动无损分页
  - **证据原则**：历史检索出的是“客观证据”，绝非“当前实时指令”

动态演示：Run 的生命周期与右移归档

系统分工：CPU 缓存 vs 磁盘阵列

记忆唤醒全流程：新会话零负担



动态演示：记忆检索与零缓存破坏折叠

除了不失忆：Agent 可以观察自己

系统设计启示

- **拒绝单一机制包打天下**：区分“会话内工作记忆”与“跨会话持久底座”
- **保护云端前缀缓存是第一准则**：
  - 记忆唤醒必须是“按需查阅 + 尾部追加折叠”
  - 严禁在 Prompt 头部随机乱插切片
- **坚固工程底座优于黑盒幻觉**：
  - 本地 SQLite + 精确工具寻址，远比盲目的 Embedding 相似度靠谱