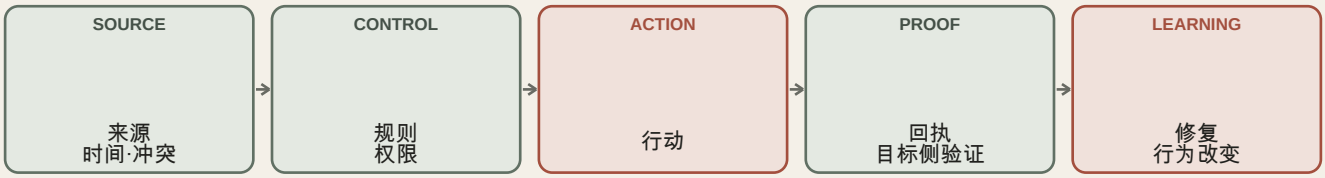
PUBLIC-SAFE FIELD GUIDE · 公开分享稿

长期记忆不是向量库

一套从来源、时间与冲突出发，闭合到行动、验证与行为改变的多 Agent 连续性设计



长期记忆不是把过去存进去，而是让一条主张能够约束行动、接受重验，并在必要时被修复、撤回或退休。

版本 2026-09-03 · 中文技术读本 · 26 页

00 · READING CONTRACT

先读证据边界

这不是产品宣传册，也不是把内部系统包装成已经完成的开源发行版。

一句话 thesis

长期记忆是一条运行合同：带来源、有效期和冲突状态的主张进入规则与权限，再通过行动、回执、目标侧验证、修复与行为改变闭环。

NOVELTY STATUS	EPISTEMIC STATUS
单独组件都有成熟先例；贡献主张是把它们组织成同一条用户可审计的运行合同。	内容混合了 synthetic invariant、单 owner 历史动机、设计 policy 与明确 open gap。
RELIABILITY	WHAT REMAINS
fixture 输出一致性高；现实行为收益、隐私边界与分布式安全仍未建立。	需要 matched held-out replay、外部复现、跨执行器目标侧验证与长期行为指标。

四个 fixture 的通过，只证明什么？

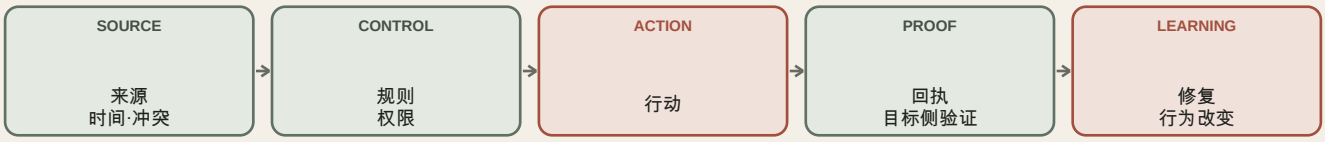
只证明预先编码的合成不变量与预期输出一致；不证明生产有效性、模型服从、隐私安全、独立验证、分布式耐久性或 exactly-once。

本稿可分享，但尚未完成外部受众冷读，因此不是“受众验证版”。

00 · SYSTEM MAP

从检索到控制：整条链究竟多了什么？

检索仍然重要，但它只回答“可能相关的过去在哪里”。连续性还要回答另外四类问题。



弧	核心问题	对应设计
A	为什么搜到了仍会做错？	constitution、raw / projection、pre-action gate
B	事实什么时候成立？	valid time、unknown / conflict、supersede、tombstone
C	多个 Agent 如何不重复、不越权、不假完成？	claim、lease、authority epoch、fence、receipt、verification
D	一次经历是否真的改变系统？	candidate、validation、promotion、rollback、behavior delta

三个最小区别

① retrieved ≠ remembered：规则若未进入动作前边界，就只是可能被想起。
② unknown / conflict、done / verified、intent / outcome 不能互换。
③ 可迁移的不是一组阈值，而是“事故 → 参数 → 反信号 → 重验”的循环。

00 · EVOLUTION

它不是一开始就长成现在这样

2026 年 4 月的出发点是 sleep-inspired consolidation；真实使用逐步把问题从“如何记住”推到“如何约束与验证”。

04 原始层 / 编译层
旅行与跨设备使用让跨会话失忆变成真实摩擦；提出离线 consolidation 与人工裁决。

05-06 从检索转向约束
关键规则不能等查询命中；形成 constitution、hot/cold 与来源回指。

07-08 从事实转向时序
引入 unknown、computed conflict、supersede、tombstone 与来源强弱。

08-09 从记忆转向控制
工作认领、authority epoch、回执、目标侧重验与 owner-stop 进入同一链。

NOW 从总结转向代谢
把 episode 变成候选变化，经验证、回滚和观察，直到下一次行为真的不同。

最重要的纠偏

早期“raw + compiled + mirror”的两层设计解决了证据与抽象的分工，却没有自动解决关键规则漏载、时序冲突、权限、并发、假完成与停止传播。后续机制都来自这些失败类型，而不是从一张理想架构图倒推。

早期公开点：LessWrong - Starshard: Sleep-Inspired Memory Consolidation for a Multi-Agent Personal System（2026-04-21）。

ARC A · RETRIEVAL → CONTROL · 01

一个 Agent 忘了，是聊天问题；多个 Agent 一起忘，是控制问题

当系统会跨 session 执行真实动作，失忆不只造成回答变差，还会造成重复执行、旧结论复活和假完成。

事故 / failure

传统 memory 常把目标写成“让模型找回过去”。但真正危险的失败发生在找回之后：谁可以行动、是否已经有人在做、当前结论是否仍有效、完成是否有目标侧证据。

机制 / design

把长期连续性定义成一条控制链。检索负责提供候选上下文；规则、权限、工作认领、回执、重验和修复负责决定系统能做什么，以及一个动作何时才算闭合。

证据状态 / evidence

合成跨 session 状态机可稳定重放两类状态：query-contingent / missing / self-report 与 pre-action / claimed / unverified。它证明状态语义被编码，不证明真实模型一定遵守。

反信号 / falsifier

若普通 thread history 或 RAG 在同一组 held-out 任务中，以更少复杂度稳定消除重复执行、旧结论复活和假完成，就没有必要引入更重的控制层。

ARC A · RETRIEVAL → CONTROL · 02

关键规则不能等 Agent 想起来再搜

只有在查询命中时才出现的规则，不是执行边界。

事故 / failure

语义检索是 query-contingent：模型先判断什么相关，系统才把内容取回。可恰恰是模型没意识到风险时，禁令、身份边界和不可逆动作条件最容易缺席。

机制 / design

将少量不可漏规则放进不可绕过的 pre-action gate。实现可以是常载 constitution，也可以是动作前的强制 policy lookup；关键不是“放进 prompt”，而是调用链无法跳过。

证据状态 / evidence

这是控制合同与可测假说，不是“always-loaded 永远更好”的断言。应比较规则召回、违规率、上下文税和误阻断，再决定某条规则属于 hot tier 还是 cold tier。

反信号 / falsifier

若 retrieval-only 在足量的 held-out 场景中漏检率并不更高，或常载规则造成更严重的干扰与误阻断，该规则应回到按需层。

ARC A · RETRIEVAL → CONTROL · 03

摘要可以重写，原始经历不能被摘要取代

一份更短、更漂亮的 memory，可能同时洗掉来源、条件和反例。

事故 / failure

如果 summary、profile 或 rule 成为唯一真相，后续系统无法知道结论来自谁、当时处于什么情境，也无法在投影错误时回放或修复。

机制 / design

保留 append-oriented raw event；compiled memory、索引、人物画像和公开文本都视为可重建投影。派生结论必须能回指 source event，投影可以更新，但不能反向改写来源。

证据状态 / evidence

合成 lineage fixture 展示 raw → compiled → redacted public projection，并验证删除投影后可从保留的源重新生成。这里的 append-oriented 不等于永久保留：仍需访问控制、保留期与受控删除策略。

反信号 / falsifier

若投影无法回指来源、重放后改变决策边界，或公开派生物混入私人原文，则证据分层失败。

ARC B · FACTS ARE TEMPORAL · 04

Hot / cold 的分界，不是“重要”和“不重要”

真正的判断是：这条信息是否必须在行动前不请自来。

事故 / failure

频率和语义相似度会偏爱常见内容，却可能压低很少被提及、但一次都不能漏的规则；把所有“重要内容”常载，又会稀释注意力。

机制 / design

hot tier 只容纳身份边界、不可漏规则、当前 owner 状态与活跃控制面；cold tier 保存历史、背景知识、长证据和可查询细节。admission 由漏检代价与干扰成本共同决定。

证据状态 / evidence

该分层来自真实使用中的反复校准，但精确大小、阈值与刷新节奏依赖具体 owner、模型和任务，不应复制成通用参数。

反信号 / falsifier

若 hot tier 膨胀后规则互相稀释，或 cold recall 经常晚于行动，说明 admission policy 需要重调；“更多上下文”不是自动修复。

ARC B · FACTS ARE TEMPORAL · 05

不知道，不等于反对；空白，不等于冲突

人物事实与长期偏好至少需要 known / unknown / conflict 三种状态。

事故 / failure

简单 key-value overwrite 会把“没找到”误写成 false，把低置信转写覆盖成新事实，或在冲突时随意选择较新的句子。

机制 / design

事实带有效时间、来源和置信信息；unknown 表示没有足够证据；conflict 由仍有效且互不兼容的证据关系计算，不能由 Agent 直接宣告。

证据状态 / evidence

确定性时序重放可产生 known、unknown、computed conflict 与 superseded。它证明 projector 的规则一致，不证明输入本身为真，也不是完整的双时态数据库。

反信号 / falsifier

若单值覆盖模型同样能保留来源、表达未知、在重放后维持一致决策边界，额外状态就没有带来净价值。

ARC B · FACTS ARE TEMPORAL · 06

别删除失败：系统会把它重新发明一遍

负知识不是垃圾；它是阻止旧路线复活的控制对象。

事故 / failure

失败实验若被删掉或只留一句“效果不好”，下一名 Agent 很容易在相似条件下重新推导、重新承诺、重新消耗同一轮成本。

机制 / design

supersede 关闭旧事实的有效期；tombstone 保存失败路线、适用范围、证据、复活条件与复验条件。它不是永久禁令，而是一张带边界的停止凭证。

证据状态 / evidence

时序原语能阻止被 supersede 的结论继续进入 current projection。是否真的减少现实重做，仍需 matched replay：同类任务中有/无 tombstone 的重复率对照。

反信号 / falsifier

如果 tombstone 造成大量假阻断，或没有减少旧路线复活，它应被缩小范围、降级为 audit-only，或退休。

ARC B · FACTS ARE TEMPORAL · 07

来源、置信度与权限必须拆开

一个肯定的小数，既不能洗白弱来源，也不能授予行动权。

事故 / failure

把 provenance 压进单一 confidence，会让重复转述看起来像独立证据；更危险的是，系统可能把“相信这句话”偷换成“可以据此发信、付费或改权限”。

机制 / design

分别记录：谁说的、原始还是转述、何时有效、系统有多确信、谁有权行动。证据更新可以改变 belief；authority 只能来自认证与授权路径，不能从概率推导。

证据状态 / evidence

当前是 schema 与治理合同；跨模型、跨媒介的统一置信尺度仍未建立。保守默认是保留来源类别与不确定性，而不是伪造精确概率。

反信号 / falsifier

若低质量来源仍能通过数量淹没一手证据，或高 confidence 自动越过权限门，系统的 belief supply chain 仍是不安全的。

ARC C · MULTI-AGENT CONTROL · 08

done 不是 verified

执行器的自报完成，只是一个需要被检查的事件。

事故 / failure

文件存在、命令返回零、进程存活或 worker 说“完成”都可能是假阳性。它们没有证明目标状态、作用范围和当前时效。

机制 / design

把 intent receipt 与 outcome receipt 分开；verification 必须是 fresh、same-scope、target-side 的读回或可验证证据。所谓“独立”还要求不同失败域，而不只是换一个 Agent 看同一日志。

证据状态 / evidence

合成 receipt checker 可将 self-report 保持为 done，并在满足读回条件后升级为 verified。它证明状态转换规则，不证明真实系统的观察源可靠。

反信号 / falsifier

若 verified 可以在没有新证据、scope 不匹配或证据过期时出现，完成语义就已失守。

ARC C · MULTI-AGENT CONTROL · 09

Liveness、authority、effect admission 是三条轴

“我看不见它”不能推导“它已经死了”，更不能推导“新 worker 可以安全接管”。

事故 / failure

断线、心跳丢失和 session 消失只说明观察能力下降。旧 executor 可能仍在运行，也可能稍后恢复并写入。把不可见当死亡会制造双写与越权。

机制 / design

liveness 描述是否可观测；authority 描述谁被允许继续；effect admission 描述下游资源是否接受其写入。三者分别记录，并通过 epoch / fence 连接。

证据状态 / evidence

三轴合成 fixture 能同时表示 liveness=unknown、authority=revoked、effect=blocked，而不把其中任何一项偷换成另一项。

反信号 / falsifier

若一个系统只靠心跳过期就重发不可逆动作，且下游没有拒绝旧 epoch 的能力，它仍可能在恢复时发生双效应。

ARC C · MULTI-AGENT CONTROL · 10

Lease 解决认领，fence 才处理迟到写

“谁应该做”与“谁还能写进去”不是同一个问题。

事故 / failure

lease 过期后，新 worker 可以接管，但旧 worker 可能并不知道自己已失权。如果效果发生处不检查 epoch，迟到写仍会被接受。

机制 / design

work claim 用于协调意图；resource-side fence 在真正发生副作用的位置拒绝 stale epoch。幂等键降低重试重复，但不能替代因果身份、权限收缩和 owner-stop。

证据状态 / evidence

单进程模拟可展示新 epoch 接管、旧 epoch 被拒绝的预期不变量。它不是生产下游资源已经实现 fencing 的证据，也不声称 exactly-once。

反信号 / falsifier

若 fence 只存在于协调数据库，真实下游仍接受旧 writer，控制层没有封住因果触达。

ARC C · MULTI-AGENT CONTROL · 11

发出去，不等于闭环

每条通道都必须回答：什么事件关闭这条循环？

事故 / failure

消息、任务、提醒或命令被“发送”只证明意图离开了发件端。中间可能丢失、重复、延迟、被拒绝，甚至已执行但回执丢失。

机制 / design

闭环通道至少包含 receipt/readback、durable state、timeout、repair/escalation、bounded retry 与最终 reconcile。owner-stop 是吸收态：旧 watchdog 或迟到结果不能悄悄重启同一授权 epoch。

证据状态 / evidence

合成 crash matrix 可区分 intent-created、delivery-unknown、effect-verified 与 repair-open。真实传输可靠性仍取决于每个 adapter 的观测与下游边界。

反信号 / falsifier

若系统无法在丢回执、重复投递和停止后迟到三类场景中给出唯一可审计终态，它仍是 fire-and-forget。

ARC D · MEMORY METABOLISM · 12

Consolidation 不是摘要

压缩旧文本不等于把经验转成可验证的系统变化。

事故 / failure

摘要可能更短，却没有回答：哪次行动产生了什么结果、哪条规则应改变、改变会造成什么副作用、失败时如何回滚。

机制 / design

consolidation 链应包含 episode → outcome → candidate → validation → promotion / rollback。候选变化可以进入 task、behavior、audit-only、tombstone 或 nothing，而不是一律写进长期规则。

证据状态 / evidence

确定性 routing fixture 能把不同证据强度映射到不同终态。路由规则目前是显式策略，不是自动学得；它不证明候选规则部署后安全。

反信号 / falsifier

若 consolidation 只能生成更流畅的总结，却无法指出下一次行为、停止条件或验证计划的变化，它仍是归档，不是学习。

ARC D · MEMORY METABOLISM · 13

记忆只有改变下一次行为，才完成代谢

存储量、召回率和摘要质量都只是中间指标。

事故 / failure

一个系统可以拥有海量 memory、漂亮图谱和高 recall，却仍在同一类任务中重复犯错，因为经验没有获得正确的 credit，也没有进入执行边界。

机制 / design

将行为改变写成可反证的 delta：下一次判断、行动、停止条件、人工介入或工具路径具体改变什么；同时记录 anti-signal、复验日期与回滚条件。

证据状态 / evidence

目前最强的证据是可重放的 candidate-routing 机制与历史动机；现实收益需要 held-out tasks、基线对照和时间窗内的重复故障率。

反信号 / falsifier

若长期存储持续增长，而 matched failures、人工重述和错误复活不降，系统只是囤积信息。

ARC D · MEMORY METABOLISM · 14

最重要的能力，是诚实标出还没闭合的器官

公开状态必须允许被降级，而不是把 roadmap 写成完成宣言。

事故 / failure

架构图、hash 相等、repo 在线或一次 demo 通过，都很容易被叙事升级成“系统已经可靠”。这会抹去 code、runtime、policy 与现实效果之间的距离。

机制 / design

每个 claim 使用多选状态：code-present、runtime-probed、enforced-on-specified-path、policy、historical、unknown。新证据可以升级；反信号必须能降级或撤回。

证据状态 / evidence

本稿包含四个 deterministic synthetic fixtures 与公开资料核对；未完成受众冷读、外部复现和生产效益评估，因此只能称公开分享稿。

反信号 / falsifier

若 claim 不能被新证据降级，或 unknown 被包装成未来必然完成，所谓可审计只是视觉标签。

EVIDENCE · SYNTHETIC FIXTURES

四个可重放工件，各自只证明一个边界

所有工件均为确定性、合成、单进程状态机；预期输出与实际输出一致。

弧	验证的不变量	明确不证明
A	关键规则在动作前出现；认领与验证状态可区分。	不证明真实模型服从，也不证明并发安全。
B	known / unknown / conflict / superseded 的投影可确定性重放。	不判断输入真假；不是完整双时态存储。
C	authority epoch、stale writer、owner-stop 与 receipt 状态转换。	fake fence 不是生产资源侧 fencing；不证明 exactly-once。
D	episode 可路由到 task / behavior / audit-only / tombstone / nothing。	策略是手写输入；不证明真实行为收益或安全部署。

为什么还值得做 synthetic fixture？

它把概念争论压成可检查的状态转换：同一输入必须得到同一投影，旧 epoch 必须被拒绝，unknown 不能偷偷变成 false。它是最低层的合同测试，不是现实世界的功效试验。

公开复验状态

代码发布仍需与公开仓库的描述和证据边界对齐。本 PDF 报告工件状态，但不把本地运行结果冒充第三方复现。

EVIDENCE · CLAIM GRAMMAR

一条 claim 应该怎样被降级？

“已实现”太粗。我们用多选状态描述它到底在哪一层成立。

code-present 存在可检查代码；不说明它被实际调用。	runtime-probed 在指定运行时通过探测；不自动外推到所有环境。
enforced-on-specified-path 某条明确路径不可绕过；路径外不作保证。	policy 行为合同或流程约定；若无结构性 gate，可能被违背。
historical 某个单系统、特定时期的观察；不能外推行业比例。	unknown 当前证据不足；缺口不是反证，也不能被包装成完成。

升级与降级规则

架构图不能升级为 code-present；代码存在不能升级为 runtime-probed；进程返回成功不能升级为 target-state verified；另一名 Agent 阅读同一日志，也不自动构成独立验证。反信号一旦出现，状态必须可降级。

可信度的来源

不是标签本身，而是标签旁边能否给出 scope、时间、来源、复验方法与失败条件。

LANDSCAPE · PRIOR ART

不是谁赢，而是哪一段链已经被谁做好

研究截止 2026-09-03；只依据公开一手文档，不推断私有实现。

公开能力	已经做得很强的部分	不要从文档缺席推断什么
ChatGPT Memory	跨对话个性化、可编辑的 memory summary、来源提示与用户控制。	不能据此断言它是通用事件账本、行动回执或多 Agent 权限控制器。
LangMem / LangGraph	hot-path 工具、抽取、后台 consolidation、持久 store 与执行编排。	应用仍需定义 provenance、冲突、权限和外部副作用正确性。
Letta MemFS	常载 system 文件与按需文件、Git 版本史、跨设备同步、并发 worktree。	Git history 不等于事实有效性、独立验证或资源侧 fence。
Graphiti / Zep	时序知识图谱、fact validity、失效历史与混合检索。	时序有效性不自动等于来源可信、用户授权或行动停止语义。

安全的组合差异说法

storage、retrieval、temporal facts、file memory 与 background consolidation 已很丰富；把 provenance、时间冲突、authority、并发、行动回执、修复和行为改变闭成同一条用户可审计链，在抽样公开材料中仍主要由分散组件与应用工程拼接。私有实现未知。

MODEL · LIVING SYSTEM

生命体隐喻，只是工程映射

代谢、睡眠、免疫、凋亡帮助我们检查闭环器官是否缺失；它们不是意识或生物生命主张。

睡眠 / consolidation 活跃期保留具体 episode；离线期做去重、冲突发现、候选规则与任务提炼。	代谢 / metabolism 经验只有经过 credit assignment、验证与行为变化，才从“摄入”变成系统能力。
免疫 / immune boundary 外来信息先保留来源与不确定性；进入规则和权限层前需要隔离、复核与最小授权。	凋亡 / retirement 失效事实、失败路线和规则不应无痕删除；用 supersede、tombstone、archive 与回滚让它们退出当前控制。
神经系统 / control plane 把感知、状态、权限、行动、反馈与停止连接起来；某个 Agent 只是可替换执行器。	稳态 / homeostasis 当证据变旧、控制丢失或 owner-stop 出现时，系统缩小因果范围，而不是靠自信维持“绿色”。

为什么这个比喻有用？

它迫使架构回答：输入从哪里来？如何被消化？错误如何隔离？失效器官如何退出？系统如何恢复稳态？若这些问题没有机制答案，比喻就应该被丢掉。

BUILD · MINIMUM CONTRACT

如果从零开始，先实现这十一个接口

不是要求一次建成巨型平台；顺序的原则是先封住不可逆错误，再优化召回。

01 · raw event append-oriented 源记录，带时间、actor、scope 与 source pointer。	02 · temporal assertion valid time、unknown、computed conflict、supersede。
03 · projection profile / index / public view 可重建，且保留 lineage。	04 · constitution gate 不可漏规则进入 pre-action、不可绕过的检查。
05 · work claim 同一 work key 的认领、期限、当前 owner 与 takeover。	06 · authority epoch 撤权、替换与 owner-stop 使用单调 epoch。
07 · effect fence 副作用发生处拒绝 stale writer；若做不到，诚实标 unknown。	08 · paired receipts 分别记录 intent 与 outcome；失败也写终态。
09 · verification fresh、same-scope、target-side readback；标明失败域。	10 · closed-loop adapter timeout、repair、bounded retry、reconcile 与 durable state。
11 · consolidation episode → candidate → validation → promotion / rollback / nothing。	PUBLIC FIREWALL 公开投影只从受控派生物生成；不从原始私人语料直接导出。

最小迁移策略

先把 done / verified、unknown / false、intent / outcome 拆开；再给不可逆动作加 pre-action gate 与 receipt；最后才做复杂的检索、图谱和自动 consolidation。

OPEN GAPS · WHAT IS NOT CLOSED

目前没有资格宣称解决的能力

这些缺口不是脚注，而是下一轮实验的输入。

- 跨所有真实下游资源的 resource-side fencing；没有 fence 时，停止后的因果触达仍可能未知。
- exactly-once 外部副作用；幂等键、lease 与重试都只能降低风险，不能魔法般提供保证。
- 跨模型、跨媒介统一的 provenance / confidence 标尺，以及来源相关性的建模。
- 自动 consolidation 的长期功效与副作用：规则膨胀、错误固化、过度抽象和身份影响。
- 独立失败域上的验证：换一个 Agent 但复用同一日志，不足以称 independent verification。
- 隐私、保留期、受控删除与公开导出的系统性评估。
- 外部复现与真实受众冷读；目前只有内部公共候选的确定性工件。

下一步最有信息增益的实验

选一组 held-out 多 Agent 任务，固定模型、token budget、工具与输入，比较 retrieval-only 与 control-chain 两臂；分别测规则漏载、重复动作、false done、人工重述、恢复时间和误阻断。所有失败都必须回写成可审计样本。

SELF-TEST · THREE QUESTIONS

你的系统是 memory store，还是 continuity layer？

不用先看技术栈，只问三道能被现场验证的问题。

01 · 事实

这条结论来自谁？何时有效？是否有仍有效的冲突证据？

02 · 行动

谁正在做？依据哪个 authority epoch？产物在哪里？谁以同 scope、目标侧证据重验过？

03 · 学习

这次经历究竟改变了下一次什么判断、行动、停止条件或人工介入？若没有，为什么仍值得长期保存？

如果三个问题里有两个只能靠人重新翻聊天记录，你拥有的可能是存储，不是连续性。

最后的判据

好的长期记忆不会让人感觉“系统记住了很多”，而是让人更少重复解释、更少确认假完成、更容易追溯错误、更能真正停止，以及在下一次相似情境中观察到行为已经改变。

REFERENCES · PRIMARY SOURCES

公开资料与研究入口

下列链接用于校验相邻能力与历史演化；产品和文档会更新，请以访问时的一手页面为准。

01	LessWrong：Starshard - Sleep-Inspired Memory Consolidation
02	OpenAI：Memory FAQ
03	LangMem：Introduction
04	LangGraph：Persistence
05	Letta：MemFS
06	Zep / Graphiti：Facts and temporal validity
07	Graphiti：Open-source repository
08	LoCoMo：Evaluating Very Long-Term Conversational Memory
09	LongMemEval：Long-term interactive memory evaluation
10	MemoryAgentBench：Incremental agent memory benchmark

关于本稿

这是一个公开安全的研究说明书，不包含私人原始记录、内部路径、运行拓扑、精确部署参数或未经授权的历史 operational fingerprints。公开代码阅读路径将在仓库描述与证据状态完成对齐后补充。

END · 记忆不是库；记忆是下一次行为仍然受到真实经验约束。