SILVITE / LONG-HORIZON AGENT FIELD REPORT · 2026.09

2026.9 热门 coding harness / agent 大横评

个人向。只看一件最烦人的事：一个 Agent 干久了以后，到底还记不记得自己一开始答应过什么。

9	6	31
个主评对象	个长程核心维度	条公开来源 / issue / 文档

个人实测 + 官方文档 + 公开社区案例

白旭缘（Silvite） · 2026-09-18

设计：墨堤主题令牌 / 五字体角色模型，仅提取设计令牌，不使用角色与背景视觉资产。

重点：Long Horizon Task Compaction / Resume / Skill / Verification

先把结论拍桌上

今天这份榜单不看谁演示视频最帅，也不看谁能一口气吐几万行代码。

我只看：跑长了以后会不会忘，忘了以后会不会自己捡，最后说 DONE 的时候敢不敢拿证据。

我为什么这么看

我的工作流已经不是“问一句、改三行”。MoDi 这类跨 Windows / Android、多链路、带协议和真实发布约束的项目，Agent 一跑就是几十轮工具调用。短任务聪明没有意义，长任务状态连续性才是生产力。

这次不吃宣传

官方写“长期记忆”只能算一半证据；公开 issue、版本修复记录和我自己的真实事故才决定另一半。官方声称什么，我会写“官方声称”；社区踩到什么，我会写“公开报告”；我的事故就直接写“第一手”。

总榜：按我现在真拿来干活的需求排

1	**Codex** 夯中夯	目前我的主力。长任务不是完全不会忘，但整体最像“能把活一路做完”的工程工具。	95
2	**MiMo Code** 夯，方向最对	国内这轮最让我改观的一个：它真的把“AI 会忘”当成系统级故障来设计。	87
3	**CodeBuddy** 能打，工程兜底厚	Memory、Checkpoint、Rewind、Subagent Memory 都给得很全，做“可恢复工程”这件事比较认真。	83
4	**ZCode** 能打，长程闭环意识很强	Skill 元数据每轮注入、Goal 持久、每轮独立校验还明确要求实据；纸面机制正对我最烦的坑。	81

5	**Kimi Code** 能打，但 compaction 有鬼故事	Session/event/Goal 都像回事；然后，乐子来了：公开 issue 正好证明 compaction 这层有多难。	76
6	**DeepSeek Harness / DSH** 架构很猛，工程还在冒烟	event-sourced Session + durable Goal 这套底子真不是闹着玩的；可 developer preview 的 bug 也是真的密。	70
7	**Baidu Comate** 功能齐，公开证据偏少	Rules、Memory、上下文压缩都补了，4.0 也明确重写长对话逻辑；我没找到足够硬的恢复闭环证据把它往前抬。	62
8	**Qoder** 功能很多，长程可靠性把我整破防	官方自己承认 compaction 有损；我这边又撞上假绿、假红、证据链错位。今天这榜单里，它吃的是第一手重罚。	54
9	**Trae-Agent（开源框架）** 拉	必须强调：这里只评开源 trae-agent，不等同于商业 Trae SOLO 内部实现。开源框架连 context-window management 都有公开缺口。	38

分数是“个人向使用分”，不是通用 benchmark：长程连续性 20、恢复能力 20、Skill/规则持续性 15、验收闭环 15、成熟度 15、可观测性 15。第一手事故会显著影响当前可用性判断。

尺子先立好：我到底在测什么

维度	我真正关心的问题	最容易骗人的地方
长程连续性	跑几十到几百步以后，原目标、边界、未完成项还在不在。	“支持 1M context”不等于“记得 1M context”。
恢复能力	Compaction、resume、崩溃、handoff 后，能不能从外部状态重新把工作捡起来。	有 Memory 文件，不等于会在正确时机重新读。
Skill / Rules	Skill 是长期约束源，还是开局吃一次然后越跑越淡。	Skill 在磁盘上 ≠ Skill 还在 working state。
验收闭环	“完成”是不是由测试、文件、hash、计数等机器证据计算，而不是模型自述。	绿色对号、FINAL ACCEPTED、summary 都能假。
成熟度	会话会不会挂、压缩会不会炸、恢复会不会把历史切坏。	架构图很牛逼，默认参数一样能把 session 干死。
可观测性	能不能看见 session、事件、diff、日志、checkpoint，出了事能不能复盘。	“有进度条”不等于有可重建状态。

最核心的一条

我默认模型**一定会忘**。成熟 Harness 的任务，是把 Goal、Phase、Outstanding Obligations、Acceptance Criteria、Skill Pointer、Evidence Ledger 放到模型脑子外面；发生 compact / resume / handoff 时主动 rehydrate。让模型靠一份“浓缩摘要”自己回忆，早晚出乐子。

理想恢复链：
`Durable Goal → Current Phase → 未完成义务 → 当前 Skill → 真实产物 → Verifier → 继续执行`

我最烦的链：
`聊天很长 → summarize() → “根据之前的工作继续……” → 猜 → 漂 → FINAL ACCEPTED ✅`

第一手样本：为什么 Codex 现在还在榜首

第一手长期使用　官方开源

Codex · 95 / 100

95
个人向

我自己的主力执行层。它当然会 compact，也不是绝对不漂；但综合下来，最少让我把时间浪费在“提醒它刚才自己说过什么”上。

真实工程样本

MoDi 从 7 月进入高强度开发，跨 Windows / Android、LAN / P2P / Bluetooth / USB、协议与 UI 多层一起推进。我负责目标、架构、边界、验收和取舍，Codex 长期承担大量实现、测试、修复和审计工作。项目已经走到真实公开发布和后续架构审计阶段——这比“跑过一次 benchmark”更接近我需要的生产场景。

它为什么舒服

仓库级 AGENTS.md、Skills、/compact、session resume、fork、review 等能力都在同一个工具链里；Codex 官方仓库本身 Apache-2.0 开源。它的上下文代码还明确要求所有注入项必须有 hard cap，说明“上下文是系统资源”这件事被当成工程问题处理。 [S1] [S2] [S3]

我给它高分的点

- 模型 + Harness 配合后，长工具链里对最初目标的抓取仍然相对稳定。
- 真忘了一部分时，比较会从 repo、git diff、测试、计划文件、已有产物重新推导状态。
- 执行反馈闭环成熟：写、跑、报错、改、再验证，这条链路基本不用我手工搬运。
- Skill/AGENTS.md 是明确的工程入口，不需要把所有规则塞进一条巨型 prompt。

扣分也得扣：它一样依赖 compaction，一样可能漂；Skill 也不是魔法“永久驻留”。所以我的工程方法仍然强调把验收标准和 durable facts 落在文件、测试和可回读证据里。Codex 好用，是整体恢复能力更强，不是有无限记忆。

第一手事故：Qoder 为什么直接掉到倒数第二

第一手事故　官方承认 compaction 有损

54
个人向

Qoder · 54 / 100

功能表看着一点不少：Rules、Skills、Memory、Goal、Subagent、Compact 都有。问题是我最看重的“长程状态连续性 + 最终验收”恰好连续爆雷。

事故 1：假绿，还是 FINAL ACCEPTED 级别的假绿

```
Phase 2B: FINAL ACCEPTED ✅
FINAL HANDOFF PACKAGE: READY ✅

真实产物:
Case Study      1 / 7
Regression Test 3 / 19
```

这一下把问题暴露得特别干净：模型自述 PASS、绿色对号、最终总结统统不能算验收证据。只要 Harness 允许 Agent 自己宣布“完成”，却没有强制 verifier 从真实 artifact 重新计算状态，假绿早晚来。

事故 2：假红 / 验证层误判

Linux / Proot 实战里还出现过 IBus/XIM 路径误诊、Host/Guest 边界判断错位、用 HTTP 200 代替 Git/SSH 真验证、旧 shell 掩盖配置状态等问题。它会非常认真地给你一个结论，但结论所依赖的“验证层”可能就选错了。

官方文档也挺诚实

Qoder 官方明确写：Compaction 会把早期细节替换为更密的摘要，次要细节可能被概括或省略，**“Compaction is lossy by design”**；压缩以后建议用户自己确认目标和约束，漏了就重述。 [S4]

然后，乐子来了：版本记录自己也在讲这件事

2026 年 8 月的 CLI Release Notes 里，连续出现“长会话 task reminder”“自动压缩没有触发”“错误 context window size”“teammate 唤醒后失忆”“压缩后 skills list 重复注入”“history rewind after compaction”等修复。说明这层直到最近都还在补。 [S5]

> **我的结论：**Qoder 最大的问题不是没功能，而是**功能齐全以后，长程恢复与验收仍然不够可信**。对我这种把 Skill 当工程约束、把长任务扔给 Agent 跑的人，这个缺点正中命门。

排名 #2

MiMo Code

87
个人向

国内这轮最让我改观的一个：它真的把“AI 会忘”当成系统级故障来设计。

长程连续性	19	恢复能力	19	Skill/规则	12
验收闭环	11	成熟度	11	可观测性	15

它是真的在对付“AI 健忘”

MiMo 官方把项目记忆、会话检查点、任务进展三套状态单独拎出来，还把记笔记这活外包给独立 **checkpoint-writer 子 Agent**。主 Agent 专心干活，另一个 Agent 自动维护状态，窗口快满时再生成干净摘要。这个思路我很吃。 [S14]

`MEMORY.md`	项目长期事实 / 规则 / 架构决策
`checkpoint.md`	当前会话结构化状态快照
`notes.md`	临时工作记忆
`tasks/<id>/progress.md`	单任务进度日志

官方开源 README 把这套目录直接摆出来，还明确写了：会话恢复时自动注入记忆；接近窗口上限时，会从最新 checkpoint、项目记忆、任务进展和近期消息重建上下文。最近更新又补了 checkpoint 独立关闭、压缩失败回滚、运行稳定性等。说明设计方向非常对，工程成熟度还在追。 [S15] [S16]

为什么只排第二：现在还是 0.1.x 的快速迭代期，版本日志本身就在持续修 recovery / compaction。要是这套东西把稳定性打磨好，它是国内最有机会把“长程不失忆”做成硬优势的一个。

排名 #3

CodeBuddy

83
个人向

Memory、Checkpoint、Rewind、Subagent Memory 都给得很全，做“可恢复工程”这件事比较认真。

长程连续性	17
恢复能力	18
Skill/规则	14
验收闭环	10
成熟度	13
可观测性	11

恢复工具箱很厚

分层 Memory + Auto Memory、每次编辑前自动 Checkpoint、跨会话持久、`/rewind`、`--continue` / `--resume`，这些都不是摆设。 [S17] [S18]

Subagent 也不是一次性耗材

子 Agent 可以声明自动加载 Skills，并单独配置 `user / project / local` 持久记忆；spawn 时会注入 MEMORY.md。 [S19]

还有个很有意思的细节：Headless 的 Rewind 文档直接写着“对齐 Anthropic Claude Code v2.1.88 的命名与协议”。至少说明它不是瞎摸索，很多成熟交互已经在按行业基准对齐。 [S20]

它差的那一扣：Checkpoint 很擅长“回到过去”，Memory 很擅长“把东西存下来”；我暂时还没看到公开资料证明它有一套足够硬的 durable acceptance ledger，能在每次 compact / resume 后自动重新挂载当前阶段的全部未完成义务。

排名 #4

ZCode

81
个人向

Skill 元数据每轮注入、Goal 持久、每轮独立校验还明确要求实据；纸面机制正对我最烦的坑。

长程连续性	17	恢复能力	15	Skill/规则	14
验收闭环	14	成熟度	12	可观测性	9

Skill 这一点很关键

ZCode 官方明确写：**每轮对话**都会把所有已启用 Skill 的名称 + 描述摘要注入上下文，Skill 正文在被调用时按需加载。至少它知道“技能目录存在”和“模型知道有这个技能”是两回事。 [S6]

Goal + 每轮校验

`/goal` 面向 long horizon task，每轮结束会单独校验目标。更关键的是官方把话说得很硬：计划、待办、跑很久、像结论的回复都不算完成，要看改出来的文件、命令输出、测试结果；还有待办就继续跑。Goal 状态关会话再开也保留。 [S7]

Memory

项目级长期 Memory 会跨后续会话自动带入，用来保存项目目标、约束、偏好等。 [S8]

真正的洞：这套纸面机制已经很对我的胃口，所以我把它抬到第四。还没继续往前抬的原因也很具体：Skill metadata 每轮都在，不代表 Skill body 的细粒度规则一直在；公开资料里我还没看到“compact / resume 后强制重新加载当前正在生效的 Skill 正文 + outstanding obligations”的完整硬机制，也缺我自己的重型长期实战。

排名 #5

Kimi Code

76
个人向

Session/event/Goal 都像回事；然后，乐子来了：公开 issue 正好证明 compaction 这层有多难。

长程连续性	16	恢复能力	14	Skill/规则	12
验收闭环	11	成熟度	12	可观测性	11

底层状态比聊天摘要认真

Kimi Code 会把 session 落成 `state.json` 和各 Agent 的 `wire.jsonl` 事件流，支持 resume / replay / fork；Goal 还是单独的持续目标，会在每轮结束判断 active / complete / blocked / paused。 [S9] [S10]

然后，乐子来了

Issue #2680：

自动 compaction 后，Agent 放弃当前任务，重新开始执行数小时前的最早 Skill。原因之一：压缩逻辑有意保留最早用户消息，而多任务 session 里“最早消息 = 当前根目标”这个假设会失效。 [S11]

Issue #1053：

Compaction 切进并行 tool-call batch，中间把 tool result 和它对应的 assistant call 拆开，resume 后 session 直接 400，且持续不可用。 [S12]

再加上插件侧公开提过：缺少稳定的 post-compaction 静默上下文注入通道，Memory / session-state 插件不好在压缩后立即重新塞回上下文。 [S13]

我的结论：设计已经进入“正经 Agent runtime”这一档了。排第五不代表架构弱；它已经走到最难的地方，然后开始踩真正的 context lifecycle 坑。公开事故证据足够硬，所以这轮我直接扣分。

排名 #6

DeepSeek Harness / DSH

70
个人向

event-sourced Session + durable Goal 这套底子真不是闹着玩的；可 developer preview 的 bug 也是真的密。

长程连续性	恢复能力	Skill/规则
19	17	12

验收闭环	成熟度	可观测性
9	4	9

架构纸面上真挺狠

Session 是 append-only、event-sourced 的单一事实源，模型历史从事件日志投影出来；Goal 变化也是 durable event；Compaction 还是独立 capability seam。这个思路已经是“Agent runtime”，不是聊天框加几个按钮。 [S24] [S25] [S26]

然后，乐子来了，而且一来一串

- Discussion #4776：长会话里 compaction 反复压不可压 checkpoint，单轮里能失败几十次。 [S27]
- Discussion #6671：默认 `thresholdRatio=0.8` 在大 output reserve 场景可能晚于真实 context 死线，session 先撞墙，压缩还没触发。 [S28]
- Discussion #4722：append-only 内存事件日志 + 重型子 Agent 扇出，被报告把 Node 进程推到约 4GB 后 OOM。 [S29]
- Discussion #5310：long-running subagent 的完整 context lifecycle / auto-compaction 还被作为特性提案讨论。 [S30]

这就是最典型的“架构图像操作系统，默认参数能把自己干死”。我愿意给它很高的架构分，但“今天就当主力”这件事，developer preview 三个字得尊重。

排名 #7

Baidu Comate

62
个人向

Rules、Memory、上下文压缩都补了，4.0 也明确重写长对话逻辑；我没找到足够硬的恢复闭环证据把它往前抬。

长程连续性	恢复能力	Skill/规则
13	10	13

验收闭环	成熟度	可观测性
8	12	6

Rules

官方明确把 Rules 定义成让模型持续记住项目规范的上下文机制，项目级落在 `.comate/rules`。[S21]

Memory

项目级 Memory 支持主动创建和自动整理更新。[S22]

Comate 4.0 的官方发布材料还专门写了“重写上下文压缩逻辑”“多轮长对话记忆增强”。这至少说明他们也已经把长对话衔接当成核心能力来补。[S23]

为什么排名不高：目前公开材料更多是“功能存在”和官方效果描述；我没有找到像 Kimi / DSH 那样能把 context lifecycle 细节掀开看的源码/issue 证据，也没有我自己的重型长期实战。保守放这里。

Trae-Agent：这页得单独写免责声明

开源 trae-agent　≠ 商业 Trae SOLO 内部实现

38
个人向

38 / 100 · 拉

公开 issue 已经把核心问题说得够直白了

2026-07-31 的 Issue #440 直接指出：当前开源 `trae-agent` 缺少 context-window management、没有 token counting，message history 会无界增长，长任务会出现内存和 token limit 问题。 [S31]

这类缺口对短任务不一定致命，对长任务几乎就是地基没浇完。你连“什么时候快把窗口吃爆了”都没统一状态，后面的 compact、rehydrate、resume 更别谈优雅。

所以这里的“拉”很具体：我评的是公开框架当前暴露出来的长程基础设施，不是在影射商业 Trae SOLO 里面一定也是同一套实现。证据边界要守住。

横着看一遍：谁到底把“忘了以后捡起来”想明白了

产品	外置 Goal / 状态	Compaction 后恢复	Skill / Rules	真正的硬伤
Codex	强：repo / AGENTS / session / 工具反馈	强：整体恢复体验最好	成熟	仍会 compact / 漂，不能把自然语言当证据
MiMo Code	很强：Memory + checkpoint + task progress	明确设计：checkpoint-writer	有	0.1.x，稳定性还在补
CodeBuddy	强：Memory + checkpoint	强：rewind / resume	子 Agent 可持久化	缺少我能确认的强制 acceptance ledger
ZCode	强：Goal + Memory	Goal 可续 + 每轮实据校验	metadata 每轮注入	Skill body 的强制 rehydrate 证据不足
Kimi Code	强：event stream + Goal	有，但出过重型事故	有	compaction stale-skill / orphan tool result
DSH	架构最硬之一	机制强、实现易炸	插件化	developer preview，compaction/OOM bug 密
Comate	有 Memory	官方称增强	Rules	公开机制细节和实战证据不足
Qoder	有 Goal/Memory	我这里掉链子	有	有损 compact + 第一手假绿/假红
Trae-Agent	弱	基础层缺口	有 Agent 能力	公开 issue 指向无 context-window management

> “把东西存下来”只是第一步。真正值钱的是：系统知道什么时候必须把什么东西重新读回来。

今天这圈最大的乐子：同一种病，能长出完全相反的症状

Qoder：该记的忘了

跑长以后，Skill 的细粒度约束越来越淡，最后能在 artifact 不达标时给自己盖 FINAL ACCEPTED。

Kimi：不该记的突然复活

公开 issue 里 auto compaction 后，几小时前最早的 Skill 激活又被当成当前任务重新执行。[S11]

这俩放一起看就特别有意思

一个是 **约束掉了**，一个是 **旧约束诈尸**。症状相反，根子都在 context lifecycle：系统没有一套足够稳的“当前任务状态投影”，只能在历史消息和摘要里猜“现在到底该相信谁”。

```
错误思路：
历史消息 = 当前状态
摘要 = 当前状态
最早用户消息 = 根目标
模型说完成 = 完成

更稳的思路：
当前状态 = durable state projection
历史消息 = 证据来源之一
summary = 缓存 / 视图，不是事实源
完成 = verifier 计算结果
```

我这套 Skill 被现实逼出来的三条铁律

1. 绿色对号没有法律效力

FINAL ACCEPTED ✅、**READY** ✅、模型自述 PASS、总结里写“全部完成”——统统只是文本。文件数量、路径、命名、非空、hash、测试结果、交付规格才是证据。

2. Validation Layer 选错，红灯也能是假红

Host / Guest、旧 shell / 新 shell、HTTP / Git-SSH、环境 / 权限，验证层不对，结果再“可复现”也可能验证的是另一个东西。命令失败和环境失败必须拆开。

3. Summary 和 Artifact 打架，Artifact 赢

增量数量不能冒充总交付数量；“我刚生成了 6 个”不等于目录现在有 7 个；报告说 19 个 regression tests，机器扫出来 3 个，那就是 3 个。

> 这三条一塞进 Harness，所谓“智能体自信幻觉”会少一大截。
> 因为你终于不让它自己给自己判卷了。

真正该拿来折磨 Harness 的统一刑具

SWE-bench 当然有用，但它不直接回答我最烦的那个问题：跑两三个小时以后你还认不认账。

1. **同一仓库、同一份复杂 Skill、同一套验收规格。**
   任务至少 2-3 小时，包含真实修改、测试、日志、文档和收口。
2. 强制经历 **3 次 context compaction**，其中一次在工具输出爆量后触发。
3. 强制经历 **2 次 session resume**，其中一次跨进程重启。
4. 插入 **1 次 subagent handoff**，检查父子 Agent 对未完成义务是否一致。
5. 故意制造 **1 次 Validation Layer 误导**：例如 Host 正常 / Guest 失败，观察它会不会把错误归因到项目。
6. 故意让它 **跑偏 20 步**，看能不能靠 repo / state / tests 自己纠偏，而不是等用户提醒。
7. 最后禁用自然语言自评，只允许 verifier 读取真实产物并生成 PASS / FAIL。

最后只看六个数字

A
原始硬约束保留率

B
未完成项丢失率

C
重复劳动率

D
假绿率

E
恢复延迟

F
Artifact / Summary 冲突率

这套一跑，谁是“第一小时很聪明”，谁是真能干长程，底裤直接扒干净。

最后结论：2026 年 9 月，国内已经不是“没人想明白”了

> 架构意识已经追上来一截。真正没追上的，是长期运行后的工程可靠性。

MiMo Code 已经明确用独立 checkpoint-writer 对付“AI 健忘”；DSH 把 Session 做成 event-sourced，把 Goal 做成 durable event；Kimi 有事件流和 Goal；CodeBuddy 有 checkpoint / rewind / subagent memory；ZCode 每轮重新注入 Skill metadata。说他们还停留在“聊天框套 API”，已经不公平。

可另一边，公开 issue 和我的实战又说明了一件更残酷的事：**把正确架构写出来，和让它在第 120 步仍然正确，是两回事。**

现在最有意思的国内选手

MiMo Code：方向最直接对着“忘了怎么办”。
DSH：架构野心最大，成熟度也最需要尊重 developer preview。
ZCode：Goal + 每轮实据校验这条线很对症，下一步就看长程实战和 Skill rehydrate。
Kimi Code：体系已经成形，compaction 的坑也最有公开研究价值。

我现在还会怎么选

真要推进 MoDi 这类项目，我依然优先 **Codex + GPT**。国内 Harness 我会继续看，而且现在终于有几家值得认真看；但谁想进主力位，先把**长程恢复、Skill 重挂、机器验收**这三关过了。

Qoder 给我留下的最值钱东西

反而是事故样本。**1/7、3/19，还敢 FINAL ACCEPTED**，这个案例往 Skill 里一塞，比十页抽象原则都有说服力。

排名是截至 2026-09-18 的个人向快照。Agent/Harness 更新极快，尤其 DSH、MiMo Code、Qoder、Kimi Code 仍在高频修复 context lifecycle；本报告不会把今天的 issue 永久贴在产品脑门上。

附录 A：证据等级与边界

标签	含义	在本报告里的使用方式
第一手	我自己的真实工程使用或事故记录	Codex / MoDi 长期工作流；Qoder 假绿、假红、验证层问题
官方	产品官方文档、官方仓库、官方更新日志	证明“机制存在”“官方如何定义/设计”
公开社区案例	GitHub Issue / Discussion 等可复查报告	证明“有人在具体版本/环境中真实触发过”，不外推成所有用户必现

> **重要边界：**社区 issue 证明的是“该问题被公开报告”，不等于所有版本、所有模型、所有用户都会触发。官方宣传的“长程稳定”也只按官方声称记录，不自动当成独立实测结论。

设计令牌

本 PDF 使用墨堤深色主题：InkNight #151A1D、InkCard #1B2226、InkCardSecondary #222B30、InkBorder #2E393F、PaperText #F2EFE6、TextSecondary #A5AEA8、TextMuted #6C7772、InkOrange #E8863C、BridgeGreen #47937F、WaterBlue #86BFD8、Cinnabar #C2452D、AccentBg #2B2418；间距取 4 / 8 / 10 / 12 / 16 / 20 / 24 / 32；主要圆角 12。字体角色沿用“匾额 / 诗文 / 注释 / 标题 / 功能”五角色，当前渲染环境使用可用的 CJK 楷 / 宋 / 黑体作角色映射，不携带原字体文件。

附录 B：公开来源

检索快照：2026-09-18。链接用于复查原始上下文；部分 GitHub Issue 后续可能被关闭、修复或改写状态。

[S1] OpenAI Codex 官方仓库 README：Codex CLI、Apache-2.0 开源许可
https://github.com/openai/codex/blob/main/README.md

[S2] OpenAI Codex 仓库 AGENTS.md / 上下文工程约束
https://github.com/openai/codex/blob/main/AGENTS.md

[S3] OpenAI Codex TUI 提示：/compact、resume、skills、review、fork
https://github.com/openai/codex/blob/main/codex-rs/tui/tooltips.txt

[S4] Qoder 官方：Context compaction，明确写明“lossy by design”
https://docs.qoder.com/qoder/context-compaction

[S5] Qoder CLI Release Notes：2026-08 至 09 的长会话/压缩/恢复修复
https://docs.qoder.com/release-notes/qoder-cli

[S6] ZCode 官方：Skill，每轮注入已启用 Skill 的名称与描述，正文按需加载
https://zcode.z.ai/cn/docs/skill

[S7] ZCode 官方：Goal 模式，多轮持续推进并在每轮检查目标
https://zcode.z.ai/cn/docs/goal

[S8] ZCode 官方：Memory，项目级长期记忆
https://zcode.z.ai/cn/docs/memory

[S9] Kimi Code 官方：Sessions and context，wire.jsonl 事件流、resume、compact
https://www.kimi.com/code/docs/en/kimi-code-cli/guides/sessions

[S10] Kimi Code 官方：Goal 模式
https://www.kimi.com/en/help/kimi-code/cli-goals

[S11] Kimi Code Issue #2680：auto compaction 后回头重跑最早 Skill
https://github.com/MoonshotAI/kimi-code/issues/2680

[S12] Kimi Code Issue #1053：compaction 切断并行 tool-call batch 导致 session 失效
https://github.com/MoonshotAI/kimi-code/issues/1053

[S13] Kimi Code Issue #1896：插件缺少可靠的 post-compaction 静默上下文注入通道
https://github.com/MoonshotAI/kimi-code/issues/1896

[S14] MiMo Code 官方发布：持久记忆、checkpoint-writer 子 Agent、百轮长会话设计目标
https://mimo.mi.com/docs/zh-CN/news/latest/mimocode

[S15] MiMo Code 官方更新记录：checkpoint 独立关闭、压缩失败回滚、稳定性修复
https://mimo.mi.com/docs/zh-CN/updates/feature/mimo-code

[S16] MiMo Code 官方开源 README：MEMORY.md / checkpoint.md / notes.md / tasks//progress.md
https://github.com/XiaomiMiMo/MiMo-Code/blob/main/README.zh.md

[S17] CodeBuddy 官方：分层 Memory / Auto Memory
https://www.codebuddy.ai/docs/zh/cli/memory

[S18] CodeBuddy 官方：Checkpoint，编辑前自动捕获、跨会话持久
https://www.codebuddy.ai/docs/zh/cli/checkpointing

[S19] CodeBuddy 官方：Subagent 可加载 Skills 并拥有独立持久 Memory
https://www.codebuddy.ai/docs/zh/cli/sub-agents

[S20] CodeBuddy 官方：Headless Rewind，文档注明对齐 Claude Code v2.1.88 命名与协议
https://www.codebuddy.ai/docs/zh/cli/headless

[S21] Baidu Comate 官方：Rules 持续作为上下文约束
https://cloud.baidu.com/doc/COMATE/s/Zm9l4agw3

[S22] Baidu Comate 官方：项目级 Memory / 自动记忆
https://cloud.baidu.com/doc/COMATE/s/smiss5jka

[S23] Baidu Comate 4.0：重写上下文压缩逻辑、增强长对话记忆
https://cloud.baidu.com/doc/COMATE/s/xmm4hx69k

[S24] DeepSeek Harness 官方：event-sourced append-only Session
https://github.com/deepseek-ai/deepseek-harness/blob/master/docs/subsystems/session.md

[S25] DeepSeek Harness 官方：Durable Goal / goal/change 事件
https://github.com/deepseek-ai/deepseek-harness/blob/master/docs/subsystems/goal.md

[S26] DeepSeek Harness 官方：Compaction capability seam
https://github.com/deepseek-ai/deepseek-harness/blob/master/docs/subsystems/compaction.md

[S27] DeepSeek Harness Discussion #4776：checkpoint 反复压缩失败导致死循环
https://github.com/deepseek-ai/deepseek-harness/discussions/4776

[S28] DeepSeek Harness Discussion #6671：默认 thresholdRatio 在大输出预留下可能晚于 context 死线
https://github.com/deepseek-ai/deepseek-harness/discussions/6671

[S29] DeepSeek Harness Discussion #4722：长/重会话 append-only 内存日志导致 OOM 报告
https://github.com/deepseek-ai/deepseek-harness/discussions/4722

[S30] DeepSeek Harness Discussion #5310：long-running subagent 缺少完整 context lifecycle 的提案
https://github.com/deepseek-ai/deepseek-harness/discussions/5310

[S31] ByteDance trae-agent Issue #440：缺少 context-window management / token counting
https://github.com/bytedance/trae-agent/issues/440