

复杂上下文控制协议的失败经验

AgentCore 窗口管理与上下文维护的实证复盘

作者: 西安博生软件 通用agent研发部

实验日期: 2026 年 10 月 5 日

整理日期: 2026 年 10 月 6 日

测试主模型: qwen3.8-flash

研究性质: 小样本工程复盘, 包含后续训练方向的研究假设

1 结论

我们的工程结论是: 在本次 qwen3.8-flash 与任务条件下, 仅通过提示词和工具接口, 尚不能让 LLM 稳定、经济地自主组合复杂上下文控制协议。 增加控制能力会增加维护成本, 也会引入模型自身难以发现的状态错误。因此, 我们当前暂不把主 Agent 自主组合窗口切换、消息折叠和前缀压缩, 作为默认的上下文管理路线。

这次尝试的初衷, 是让 Agent 像人整理工作材料一样, 将阶段工作分开, 在需要时恢复旧阶段, 用摘要替换不再需要逐字保留的过程。协议确实能够改变模型看到的消息, 底层工具也完成了窗口隔离、折叠、压缩与原文回读。但“能够控制上下文”没有自然转化为“能够稳定、经济地控制上下文”。

第一项代价是累计 token 消耗增加近五倍

在两次相同三轮任务的对照记录中, 复杂协议组累计消耗 103,861 tokens, 普通历史基线为 17,399 tokens。复杂协议组总量约为基线的 **5.97 倍**, 增量约为 **497%**, 即增加近五倍。复杂协议组的核心任务通过数为 1/2, 基线为 2/2。这里统计累计输入与输出, 包含首次失败后的诊断续跑; 这不是单次窗口长度、计费金额, 也不是已证明的普遍节省或损耗比例。成本口径与敏感性分析见第 2 节。[1]

第二项代价是信息恢复依赖模型识别自己的遗漏

复杂协议使模型需要同时判断业务下一步、当前窗口、要保留的证据、摘要范围和回读时机。错误操作可能覆盖工作记忆, 或把关键证据排除在后续视图之外。虽然原始消息仍然存在, 模型却未必知道自己缺了什么, 也就未必会调用回读工具。

****我们把这种风险称为自主执行中的有效信息丢失。其“不可逆”指: 如果可见上下文连发现遗漏所需的线索也丢失, 仅靠当前模型的后续决策, 不能保证恢复原有信息; 它不等于底层日志被物理删除。实测确认了误覆盖、局部历史被误认为完整历史和多份记忆冲突; 尚未严格证明所有信息缺失都无法被模型恢复。**

这一结论不否定方案的未来可行性。 一种值得验证的解释是: 当前测试模型对如何管理自己的上下文、如何识别信息缺口, 还没有形成足够稳定的策略。面向该协议的监督训练或强化学习, 有可能提升工具选择、信息保留与自主恢复能力; 这些路径并未在本次实验中验证, 不能承诺一定解决问题。

本文是一份小样本工程失败复盘。它支持我们调整当前模型的使用方式, 不构成“所有 LLM 都无法使用复杂协议”或“未来经过训练的模型也无法胜任”的普遍命题。

2 我们做了什么 实测效果如何

2.1 测试模型与实验条件

本次测试使用的主模型是 **qwen3.8-flash**。这是 PPE 中登记并实际调用的模型名称；应用配置通过 default 选择该模型。最后一轮消息映射实验与三轮任务对照的推理记录使用同一模型定义和同一模型账户，没有在实验组与基线之间切换模型。[1]

项目	实验条件
运行环境	AgentCore PPE，通过 MCP 启动并检查真实模型执行
测试日期	2026 年 10 月 5 日
主模型	qwen3.8-flash
协议版本	unified@1、unified@2 的探索测试；unified@3 的消息映射与多轮对照
最后一轮对照基线	prefix@2 普通会话历史，不向主模型暴露手动上下文管理工具
最后一轮样本	三种映射各 3 次冷启动，通过组最多 2 次工具续接；多轮两组各重复 2 次
多轮运行限制	每个 Run 60,000 自身 tokens、90,000 调用树 tokens、12 次推理、10 次工具调用；实际预算行为见 2.8 节
本次模型训练	未针对协议进行监督微调或强化学习；实验调整的是协议、提示词、工具约束及消息映射

早期主动压缩测试调用了应用内已发布的维护子 Agent 模板。最后一轮用于成本对照的多轮任务没有触发维护子 Agent，其消耗来自主模型。本文不把早期维护子 Agent 的全部配置等同于主模型配置，也不将一个主模型的结果外推到所有基础模型。[1][2][3]

2.2 将三类能力合并成独立协议

我们在 AgentCore 中实现了独立的 unified 上下文协议，按“窗口在顶层、折叠在窗口内、前缀压缩处理剩余历史”的层次组织能力。本次实现不包含压栈出栈，也未引入监控 Agent 或历史经验路由。三种能力是条件触发的选择，不是每个任务都要依次执行的流程。[4]

能力	模型可以做什么	对后续可见上下文的影响
窗口管理	创建、切换、读取窗口；保存阶段记忆与成果	主要展示当前窗口过程；其他窗口历史按需读取
消息折叠	查询真实消息范围；将完整交互组替换为简述	用折叠摘要代替细节；原文保留并可分页回读
前缀压缩	调用维护 Agent 整理阶段前缀；宿主也可按容量触发	用窗口检查点承接已覆盖前缀；不自动恢复旧前缀

模型还要维护跨窗口的公共任务记忆，以及各窗口的局部记忆、交接输入、成果和检查点。提示词要求：简短任务直接完成；独立阶段才切窗；完整且冗长的片段才折叠；剩余历史仍有维护需要时才压缩。

2.3 原始消息如何成为模型消息

原始消息按会话顺序保存，记录所属窗口；切窗不重新归属旧消息。每次推理前，宿主冻结当前窗口、状态版本和消息上界，从当前窗口提取尚未被前缀压缩覆盖的历史，再合入当前 Run 的受保护用户原文，按消息身份去重并排序。有效折叠范围由带来源引用的简述替代。

基础映射为：**系统提示 → 上下文头 → 选中的原始历史及折叠简述**。上下文头包含公共约束、当前窗口、局部记忆和阶段检查点。后续版本增加“当前执行位置”派生状态块，明确窗口 ID、版本和交接状态。派生块不写回原始会话，也不伪装成工具结果。

工具调用与结果保持配对。切换窗口的调用和结果归属于离开的窗口，下一次推理才使用目标窗口。恢复旧窗口时，历史末尾可能仍保留过去“离开该窗口”的动作；模型必须区分历史事件与当前执行位置。

2.4 机制约束能保护什么

折叠不允许跨窗口、重叠或拆开工具调用与结果，也不折叠用户输入和未完成交互。折叠没有取消操作，但提供原文回读。前缀压缩保留原始记录，并维护覆盖边界。v2 又增加 expectedWindowId：模型声明的当前位置与宿主不符时，拒绝写入。

这些约束保护消息结构、操作范围和状态一致性，但无法证明模型写入的业务内容正确。**窗口 ID 正确，不代表记忆内容正确；工具返回成功，不代表本次上下文决策正确。**

2.5 先验证工具机制 再验证模型使用行为

我们分开检查两件事：一是工具是否按契约修改视图，二是模型能否在业务任务中合理使用工具。早期 PPE 测试在明确要求执行相关操作的条件下，验证了窗口隔离、折叠替换、主动前缀压缩，以及压缩后仍可读取折叠原文。原文中的证据标识 ALPHA-EVIDENCE-6913 在压缩前后均可回读。[2][3]

同一批探索性测试也暴露了恢复问题：v1 恢复报价窗口 alpha 后，模型把排期窗口 beta 的内容写入 alpha，实际覆盖了报价工作记忆。加强提示词后仍出现同类错误意图；该次调用因预算限制未执行。v2 的窗口校验成功拦截了位置不匹配的写入，但恢复后重复打开窗口、重复回读和误解执行范围的问题仍然存在。[2][3]

2.6 控制消息映射实验

为检验异常是否主要来自派生状态的位置或角色，我们增加实验性 unified@3。三组使用相同的中性提示词、工具 Schema 和状态文本，仅改变状态块的位置或角色；实际请求快照与哈希用于核对条件。主模型为 qwen3.8-flash，使用同一模型定义与账户。

组别	状态块映射	冷启动算术题	工具续接
A	历史末尾 assistant	3/3 通过	2/2 通过
B	历史之前 assistant	3/3 通过	首次漏答余额后停止
C	历史末尾 user	3/3 通过	2/2 通过

冷启动题要求直接计算 4,200 + 5,300，预算 9,800；正确结果是总价 9,500、余额 300、未超预算。工具续接题要求仅调用一次窗口列表工具，再给出指定完整答案。B 组工具本身成功，但最终漏掉余额，未通过完整答复门槛。

A 与 C 均通过五项小测试，不能断言“末尾 assistant 必然导致错误”。C 按事前规则以较低累计 token 进入多轮测试，但只比 A 少 162 tokens，不能视为稳定的效率优势。旧 v2 的附加哨兵测试两次中有一次声称没有任务；v2 与 v3 的提示词也不同，该结果不能隔离角色或位置的唯一因果。[1]

2.7 用真实三轮会话检验约束更新

多轮实验比较 C 组与 prefix@2 普通历史基线，各重复两次。两组使用相同业务输入、基础任务提示、模型与运行限制。基线不向主模型暴露手动上下文管理工具，保留普通会话历史；短任务没有触发自动压缩。它并不是一个已验证的自动阶段路由系统。

第一轮核对两项报价，总价 9,500，预算 9,800，并保存结论；第二轮登记交付日期 2026-10-12、负责人林舟和凭据 SCHEDULE-2086；第三轮把预算正式改为 9,000，要求保留原报价、明确旧预算失效、计算超出 500，并复述排期。全程禁止外部写入、下单与付款。后两轮续接同一根会话，而非三个独立任务。

2.8 多轮结果与成本

方案与重复	核心任务结果	推理次数	工具次数	累计 tokens
复杂协议 第一次	失败	6	3	39,757
复杂协议 第二次	通过	11	8	64,104
普通历史 第一次	通过	3	0	8,299
普通历史 第二次	通过	3	0	9,100

复杂协议两次合计 103,861 tokens，基线合计 17,399。按这组完整执行记录，总量之比约 5.97，增加约 497%。第二次双方都完成任务，复杂协议消耗约为对应基线的 7.04 倍。所有数字均为实际累计输入加输出，未按缓存价格折算；多轮读数没有重复累加。[1]

****诊断续跑的影响。****复杂协议第一次在第二轮已判失败，但测试编排在审阅该答复前已发出第三轮。第三轮只作诊断，未挽回判分，却贡献了 16,715 tokens。若从复杂协议总量中扣除它，剩余 87,146 tokens，仍为基线的约 5.01 倍；此时该组少完成一轮任务，不能当作等量成功工作的比较。增加约 497% 是完整记录口径下的描述，不能解释成协议本身的纯因果效应。

****预算口径。****原计划把 60,000 token 限制描述为每轮上限，实测接口复用同一 Run，实际约束整段会话。两组保持相同冻结配置。复杂协议第二次在最后一次推理前累计 57,448，最后调用再用 6,656，得到 64,104；运行时允许保留这次正常结束答复。业务通过不意味着满足严格的 60,000 token 成本上限。

2.9 两个最有解释力的失败现象

****任务消息存在，模型仍漏答。****复杂协议第一次第二轮只输出空的 Agent Inputs 块，没有报告排期。实际准备发送给模型的请求中包含完整排期消息，因此不能把这次错误归为传输时删除了用户输入。它说明模型即使获得任务，也可能把注意力与输出转向协议元信息。

****最终答案正确，派生记忆仍不一致。****诊断第三轮中，模型正确回答新预算下超支 500，并补存排期，但只更新窗口记忆。公共任务记忆仍写“预算上限 9800 元”“长期有效”，同一请求里的窗口记忆却写“9000 元当前有效”。下一次只依赖某一份摘要时，旧约束可能再次变成判断依据。

第二次重复没有出现上述漏答：模型创建了排期窗口，又搜索并创建新的报价复核窗口，最终同步更新公共记忆。但它没有恢复原报价窗口，且用了八次工具调用。两次路径不同，说明一次成功不足以证明稳定的阶段管理策略。

基线也有措辞和冗长问题，例如把题面中归属有歧义的两项都写成供应商 A；核心通过不代表输出完美。本轮多轮用例没有触发折叠或压缩，不能据此评价它们在长历史中的收益。

3 失败机制 设计启示与未来可行性

3.1 上下文管理本身成为了额外任务

每一次上下文工具调用都需要模型读取工具定义和当前状态，生成参数，等待结果，再发起下一次推理。公共记忆、窗口记忆、交接与摘要中的重复内容，也会被再次输入。于是一次业务计算被扩展成一串管理决策，累计成本可能上升，即使某次推理的可见窗口确实变短。

早期 v2 功能验收提供了一个具体例子：一次主动压缩前后，父任务单次输入从 8,342 降到 6,365 tokens，而维护子 Agent 消耗 44,262 累计 tokens。[3] 这个观测不能直接用于计算长期盈亏平衡，因为维护前后的输入还包含新增交互；它足以说明，窗口缩短与总成本降低必须分别测量。

3.2 错误管理会改变模型后续判断所依据的信息

普通业务工具的结果通常向上下文增加信息；上下文控制工具还会改变哪些历史能继续被看到。模型一旦选错窗口、误覆盖记忆或生成不完整摘要，后续推理就可能以错误的状态表示为起点。被遗漏的事实和“这份摘要有所遗漏”的线索，可能一起退出可见范围。

早期 v2 测试中，模型把同一 Run 在其他窗口发生的操作描述成“非本轮新调用”，尽管原始时间线完整。[3] 这直接暴露出执行范围认知偏差：模型将当前窗口呈现的子集误认为完整过程。v3 的新旧预算冲突则表明，即使没有切窗，多份派生记忆也可能产生不同的当前事实。

3.3 能回读原文不等于能发现该回读什么

自主恢复至少需要三个条件同时成立：原始记录仍可访问；模型意识到当前信息不足或互相矛盾；模型能定位相关原文并正确纳入当前判断。提供读取工具只直接解决第一个条件，并不能保证后两个条件。

例如，某段原文包含一条例外约束，摘要只留下普通规则。如果后续视图没有保留例外标识、证据索引或冲突提示，模型可能根据普通规则顺利继续，既不报错，也不主动检索。这是**机制上的风险示例**，不是本轮已观测到的折叠丢失案例。当前多轮实验中的用户原文仍跨窗口可见，因此还不能把其失败归因于折叠或压缩造成了原文缺失。

这也限定了“不可逆”的含义：对存储层而言，原文仍在，可以被人工或宿主取回；对自主决策而言，若触发恢复所需的线索不再可见，系统就没有可靠保证能自行回到正确状态。永久不可恢复尚未得到证明，但“有回读工具即可安全遗忘”的设计假设已经缺少依据。

3.4 工具结构校验不能替代语义校验

expectedWindowId 能拒绝“模型以为在 beta，实际上在 alpha”的写入，却不能拒绝“模型知道在 alpha，却把 beta 的内容写进来”。同样，版本号能防止过期状态提交，不能识别摘要是否丢掉例外、公共预算是否已经过时。我们需要把工具成功率、信息保真度、任务正确性和总成本分别评价。

3.5 我们会如何收缩下一版设计

****首先减少主模型必须作出的管理决定。***将分页、完整交互分组、容量检查、调用结果配对和可机械判断的维护交给宿主。让主模型表达“本阶段完成”“需要恢复某项证据”等少量业务意图，而非持续手工维护多套上下文状态。这是下一步设计方向，尚未通过本轮实验验证。

****为当前约束指定唯一权威来源。***用户变更预算时，应有明确版本和旧值失效关系。窗口摘要可以保存“当时按 9800 计算”的历史结论，但不能继续把它标为当前硬约束。宿主应尽量暴露结构化的更新与冲突信息，减少依赖模型同时改写多份自由文本记忆。

****保留能发现遗漏的索引。***仅保存原文地址还不够。可见状态应保留阶段清单、证据引用、摘要覆盖范围及未解决问题。执行关键结论或外部动作前，可以由宿主要求检查指定证据，避免完全依赖模型自发意识到缺失。恢复机制需要独立测试，而不是根据“原文读取接口返回成功”推定成立。

****把窗口收益作为可选能力的准入条件。***短任务优先保留普通历史。只有在足够长的任务中，阶段隔离带来可复查的正确性或总成本收益，才增加窗口控制。是否折叠、是否压缩，应由实际信息体量与后续使用需要决定，不能将管理动作变成阶段结束时的固定仪式。

3.6 未来可行性与专项训练方向

我们仍然认为，自主上下文管理存在可行性。当前失败说明这一能力不能仅靠暴露工具和补充提示词来假定成立，并不意味着经过专门训练的模型也无法掌握它。更合适的后续问题是：模型能否学会评估一次上下文操作对未来任务的影响，并在质量与累计成本之间作出合理选择？

3.6.1 一种解释是模型尚未充分学会管理自己的上下文

我们的工作假设是：本次使用的基础模型，对“哪些信息会在切窗后不可见”“哪些约束必须跨阶段保留”“何时应怀疑摘要不完整”等问题，尚未形成足够稳定的行为策略。理解工具参数与理解工具对未来信息状态的影响，是两种不同要求。

本次实验通过提示词告诉模型规则，但没有针对协议进行参数训练。仅凭这些结果，不能确认模型原有训练中是否包含类似能力，也不能证明训练不足就是唯一根因；协议设计、消息映射、任务长度与推理预算同样可能影响结果。因此，“模型理解不够深”是值得验证的解释，而非已经由本次实验确立的因果结论。

3.6.2 用监督训练建立正确的管理行为

一种可行的起点，是构造包含完整任务轨迹的监督训练数据：展示哪些阶段值得隔离，哪些短任务应直接完成，哪些证据必须留在公共约束中，以及恢复旧窗口后应如何避免重演历史动作。训练样本还应包含错误后的修复，而不仅是从头到尾顺利的调用序列。

尤其需要教会模型识别“我现在看到的只是历史子集”。训练可以要求模型保留证据索引、区分当前约束与历史结论，并在存在冲突或证据不足时定向回读。目标是学会维护可继续推理的信息状态，而不是记住固定的窗口、折叠、压缩调用顺序。

3.6.3 用强化学习优化长期结果与恢复能力

强化学习是值得尝试的下一条路径，可能改善甚至解决当前观察到的部分能力问题，但本次尚未验证其效果。上文控制动作的影响往往在后续轮次才出现：一次压缩当时节省了输入，却可能在稍后丢掉例外条件；一次额外回读当时增加了成本，却可能避免最终错误。训练目标需要覆盖整段执行，而不能只奖励当前工具调用成功。

可以在保留完整原始日志的训练环境中，让评估器检查以下结果。完整日志用于训练与评估，执行模型仍只看到协议实际提供的视图，避免把恢复答案直接泄露给模型。

训练目标	可观察的评价信号	需要避免的投机行为
完成业务任务	最终答案、硬约束和证据一致	用流畅回答掩盖证据缺失
保持当前状态正确	预算等约束更新后，当前记忆同步且旧值明确失效	仅改最终答案，保留冲突记忆
自主发现并修复信息缺口	无人提示下识别矛盾、定位原文并恢复正确结论	等待评估器指出具体遗漏
控制端到端成本	在任务正确的条件下，减少累计 token 与无效调用	为省 token 过度压缩或跳过核验
按需使用控制能力	短任务不管理，长任务在有收益时管理	为获得过程奖励而机械调用工具

训练时应优先满足任务正确性与关键约束，再优化成本。单纯奖励压缩率可能鼓励模型遗漏信息；单纯奖励回读次数又可能导致把全部原文反复读回。有效的策略需要学会何时管理、保留什么、何时恢复，以及何时直接继续业务工作。

强化学习也不能替代原始记录保留、状态版本检查和明确的工具契约。即使模型经过训练，运行时仍应保护恢复入口，并为关键约束提供独立于自由文本摘要的检查手段。

3.6.4 如何判断训练是否真正解决了问题

后续应在冻结协议与相同评测条件下，比较原始模型、仅优化提示词的模型、经过监督训练的模型，以及进一步经过强化学习的模型。普通历史基线也应保留，避免只证明训练后的模型优于自身旧版本，却没有证明复杂协议具有净收益。

评测需要包含未见过的任务、不同历史长度、多轮约束变化和可控的信息遗漏。重点观察任务正确性、自主缺失发现率、正确恢复率、记忆冲突和累计 token。只有在未见任务上稳定改善这些指标，才能说明模型学到了可迁移的上下文管理能力，而不是记住了几条测试路径。

因此，我们对这一方向的判断是：**近期采用较少的自主控制和较明确的宿主规则；长期保留通过专项训练，尤其是强化学习，提升模型自主上下文管理能力的可能性。**当前模型的表现决定当前部署策略，不应被当作未来模型能力的上限。

3.7 证据边界与后续验证条件

本次只使用一个主模型，映射组各有三次冷启动、最多两次工具续接，多轮每组只有两次重复；没有统计显著性结论。复杂协议与基线同时存在提示词、工具和投影方式差异，无法把结果归因于单个因素。早期功能验收显式要求调用折叠与压缩，不代表模型自然选择时机正确；长历史自动容量维护、跨模型泛化和完整自主恢复仍未验证。

如果继续测试，我们会先定义“关键约束是否被保留”“是否能主动发现缺失”“能否在无人提示下找回证据”等判据，再与普通历史基线比较相同任务的端到端成本。必须把恢复提示是否来自测试者单独记录，否则人工指出缺失后成功回读，会高估系统的自主恢复能力。

这次失败改变的是我们的默认假设：给模型更多上下文控制工具，并不等于模型已经学会了管理上下文。当前应依靠清晰的宿主规则保护信息一致性；未来可以通过专项训练检验模型能否逐步承担更多控制责任。自主上下文管理仍是一个值得研究的方向，其价值需要由正确性、恢复能力和端到端成本共同证明。

实验记录与复查入口

[1] [unified@3 消息映射与三轮任务对照报告](#)，2026-10-05。附[预注册实验计划](#)与[冷启动请求哈希](#)。公开版保留实验设计、哈希核对及结果汇总，不提供逐轮原始消息、实际请求和推理快照。

[2] [unified@1 PPE 首轮探索测试报告](#)，2026-10-05。公开版保留功能、提示词加强与自主任务的结果汇总及分析。

[3] [unified@2 PPE 复测报告](#)，2026-10-05。公开版保留窗口校验、原文回读、短任务和恢复范围认知的结果汇总及分析。

[4] [AgentCore 综合上下文协议设计说明](#)。描述原始消息到模型视图的映射、窗口与折叠状态、压缩边界，以及 v1 至 v3 的设计变化。

以上为项目内实验记录，链接采用相对于本文的路径。本文为工程经验复盘，未作独立同行评审；未来训练方向为研究提案，不是已完成实验的结果。