Morphz：结构化上下文上的非确定性认知符号求值

Raymond Ren

英文题目：**Morphz: Nondeterministic Cognitive Symbol Evaluation over Structured Context**

摘要

现有语言模型智能体通常把认知状态隐含在追加式消息、提示词和工具循环中，缺少跨会话稳定的对象身份、修订关系与结果回流路径。本文提出 Morphz：程序、认知数据和求值结果采用同一种可递归嵌套的结构化表示，大语言模型则结合智能体自有的结构化上下文，对其中的认知符号进行**非确定性认知求值**。认知帧、关系、观察对象、绑定、来源和版本构成可寻址的持续计算状态；模型返回值可以进入当前表达式，也可以成为后续任务直接引用的持久认知对象。模型返回的候选 Yao 程序还可以在输出类型、有效副作用集合与权限校验后成为一等程序值，经按求值所有者分派的持久子执行把结果接回原计划。确定性运行时负责程序校验、权限、版本、真实观察记录与权威状态提交。

分层实验表明，该机制能够在三种等信息程序表示和九种模型配置上执行；九种模型的非确定性求值中，104/108 个响应返回了可解析且符合可见契约的选择值，8 项权威状态提交与故障处理测试全部通过。在 STATE-Bench 三系统对照中，Morphz、Letta 和 Memo 支撑的参照系统均使用 GPT-5.6 Sol/max；Morphz 在 150 个未参与训练的测试任务上通过 122 题（81.33%），Letta 通过 93 题（62.00%），Memo 支撑的参照系统通过 96 题（64.00%）。轨迹审计确认全部 Morphz 测试任务都使用了对应领域由 100 条训练轨迹形成的第 100 版结构化上下文，三个领域共计 300 条训练轨迹。在同模型、同主机和同冻结协议的 Terminal-Bench 2.1 完整评测中，最新修复版 Morphz 与 Codex CLI 0.149.1 均为 74/89；差值的 95% 自助法区间为 [−7.87,+7.87] 个百分点。结果支持结构化认知状态、非确定性认知求值、经验迁移，以及由确定性运行时控制权威状态与现实副作用的机制；S 表达式只是当前实现载体。

关键词：大语言模型智能体；非确定性认知求值；认知符号；结构化上下文；S 表达式；持久认知状态；确定性提交

术语说明：本文首次出现时采用中英文对照，后文统一使用中文：智能体（Agent）、结构化上下文（Structured Context）、认知帧（Mind Frame）、观察对象（Observation）、上下文事务（Context Transaction）、运行时（Runtime）、会话（Session）、线程（Thread）和主体（Principal）。Morphz、Yao、模型名称、公开基准名称及代码标识保留原文。

1. 引言

大语言模型把大量世界知识、语义联想和不完备信息推理压缩在参数中，使系统能够对没有被程序员穷尽定义的符号进行有用判断。ReAct、Toolformer 等工作进一步表明，模型可以在推理过程中选择行动并利用外部工具 [2,3]。然而，主流智能体的计算本体仍大多是“对话消息 + 提示模板 + 工具循环”：历史以文本追加，状态以临时提示或外部记忆片段回填，中间结果往往只作为消息中的文本片段或一次性结构化输出返回，没有成为具有稳定身份、类型、来源和引用路径、可被后续计算直接使用的状态对象；下一步仍依靠模型重新阅读和解释这些内容。

该范式存在四个相互关联的问题。

第一，线性文本历史弱化了对象身份和关系。一个目标、一次观察、一个用户偏好和一条已被修订的事实在文本上都是段落；系统很难以统一方式表达“派生自”“取代”“保护”“属于某个线程”或“只在某一局部作用域有效”。

第二，计算结构与认知数据相互割裂。工作流、工具计划或 Python 程序通常位于控制层，记忆、知识和中间结论位于数据层。模型可以生成程序，也可以读取记忆，但二者通常不属于同一可递归操作的表示域。

第三，中间认知结果缺少稳定的回流路径。模型的一次判断可能影响下一轮提示，却很少以具备身份、来源、版本和引用路径的结构化值重新成为后续计算的操作数。这使长程推理依赖反复的自然语言重解释。

第四，非确定性认知候选与已提交的权威状态常被混在同一工具循环中。模型既提出认知判断，又直接发起写文件、发送消息或修改权威状态；系统没有明确表达哪些内容只是候选认知值，哪些变化已经成为现实。缺少这一区分时，模型的认知弹性和运行时的可验证性都难以成为稳定的系统性质。

Morphz 的基本假设是：智能体不应以对话作为计算本体，而应拥有可版本化的结构化上下文，并由大语言模型在这个上下文和当前观察对象下对认知表达式进行非确定性求值。程序、认知对象和返回值共用同一种递归表示，使结构化结果能够进入当前绑定或后续上下文，而不是退化为只能重新阅读的文本。CPU 执行语义已经完全定义的确定性计算，运行时负责程序校验、权威状态提交和现实副作用，为这一认知计算提供验证、提交和恢复边界。

本文研究以下问题：

> *能否把大语言模型组织为结构化、持久上下文上的非确定性认知符号求值器，使程序、认知数据和返回值共用同一种递归表示，同时让权威状态提交和对外部系统产生的现实副作用继续由运行时控制？*

本文作出四项贡献：

1. 定义大语言模型的一种计算身份：在结构化上下文、观察对象和显式算子求值内核的共同条件下，充当非确定性认知符号求值器。
2. 把结构化上下文定义为持续求值状态，并让认知对象、候选程序和返回值共用同一种递归结构化表示，使它们可以被寻址和消费，而不是只以文本追加；先前任务还可以经事务整合形成或修订认知帧，并把这种显式经验作为后续会话的计算状态参与求值。
3. 描述 Morphz 的工作实现：采用 S 表达式语法的 Yao 语言、带明确类型的结果绑定、可接受以 `eval` 或 `infer` 为根的 Yao 程序值、按求值所有者分派的持久子执行、版本化上下文事务、确定性计划验证与提交、持久事件记录和可恢复调度器。
4. 通过受控机制实验、九种模型配置测试、STATE-Bench 和 Terminal-Bench 系统对照，评估上述机制的可执行性、跨模型适用性、经验迁移能力和系统级任务表现，并报告其失败模式与适用边界。

2. 背景与相关工作

2.1 符号表达、程序与数据

Lisp 的关键历史贡献之一，是用递归列表统一表示符号结构，并使程序本身能够作为数据被读取、构造和变换 [1]。这种同像性（homoiconicity）降低了程序与数据之间的表示边界。但传统解释器仍要求算子的语义由规则完整定义。例如 `(+ 1 2)` 可以由 CPU 稳定求值，因为 `+`、数字和求值顺序均有明确规则；而下列表达式无法由传统机器直接处理，除非开发者预先穷尽“冲突”“证据”和“偏好”的语义：

```
(resolve-conflict
  (experience old)
  (evidence new)
  (preference user))
```

Morphz 延续的是“知识、程序和数据可以采用同一种递归符号结构”的思想，而不是声称 S 表达式语法本身构成创新。不同之处在于，大语言模型提供了一种非确定性认知求值器：它能结合自然语言说明、结构位置、上下文、观察对象和参数先验，对未被完全形式化的认知符号产生条件语义值。

2.2 语言模型与程序执行

Program-Aided Language Models（PAL）和 Program of Thoughts（PoT）让模型生成 Python 等程序，再由外部解释器获得精确结果 [4,5]。BINDER 将语言模型能力绑定进 SQL/Python 等符号语言：模型既生成程序，又可在程序执行过程中通过语言模型接口处理传统符号语言覆盖不了的语义，接口输出以变量形式继续参与执行 [6]。LMQL、SGLang 和 DSPy 分别从约束查询、结构化生成和声明式优化角度把模型调用提升为可编程对象 [7–9]。

这些工作表明了“模型与程序语言交错”以及“模型输出继续参与程序”的价值，也是 Morphz 最接近的技术前序之一。Morphz 与它们的主要差异不是是否调用大语言模型，而是计算对象的范围：BINDER 主要把语言模型调用嵌入任务级符号程序；Morphz 试图让持久认知状态、认知操作、候选程序和返回值属于同一递归表示域，并让结果跨轮次、跨会话进入智能体自有的结构化上下文。

PLSemanticsBench 直接测试大语言模型能否依据给定的形式语义充当传统程序解释器。其结果显示，模型在标准语义上表现较好，但在非标准语义和复杂执行轨迹上仍不稳健 [10]。该问题与本文相关但不相同：Morphz 不要求大语言模型精确模拟传统程序语言中由规则完全定义的每一步操作语义；确定性部分可以交给运行时。本文关注的是传统解释器无法穷尽定义的认知符号如何被结构化表达并进行非确定性求值。

2.3 工具智能体与受约束执行

ReAct 以交错的推理和行动轨迹连接模型与环境 [2]；Toolformer 探索模型自主决定何时调用 API [3]；Reflexion、Voyager 等系统将反馈或可复用技能用于后续任务 [11,12]。这些工作展示了模型形成行动闭环的能力，但其主要认知载体仍是自然语言轨迹、反思文本或代码技能。

ToolGate 使用显式符号状态以及类似 Hoare 逻辑的前置/后置条件，为工具执行提供可验证状态演化 [13]。它与 Morphz 的运行时提交边界高度互补：两者都拒绝把模型输出直接视为已提交的权威状态。但 ToolGate 主要解决工具调用的逻辑安全，Morphz 的核心问题还包括认知状态的表示、非确定性认知求值以及结果如何成为后续认知项。

2.4 智能体记忆与认知架构

Generative Agents 使用观察流、反思和检索支持长期社会行为 [14]；MemGPT 通过分层记忆和虚拟上下文管理有限窗口 [15]；CoALA 将语言智能体组织为包含工作记忆、长期记忆、决策过程和学习过程的认知架构 [16]；AIOS 从操作系统角度管理智能体的调度、上下文、记忆和工具 [17]。

2026 年的 Agentic Memory 把存储、检索、更新、摘要和遗忘暴露为智能体可学习的动作 [18]；StructMem 进一步研究结构化事件记忆对长程行为的作用 [19]。这些工作强化了“记忆应由智能体主动管理”这一方向。Morphz 的差异在于：上下文不只是一组可检索记忆，而是程序求值的当前状态；认知帧、关系、观察对象和表达式值通过显式引用、修订和事务关系参与计算。换言之，记忆管理是 Morphz 结构化上下文上的一类程序，而不是一个独立外挂模块。

这一差别也改变了“学习经验”的含义。迁移不只是保存旧任务回合，并在相似问题到来时检索一段文本；先前任务可以促使智能体形成、修订、关联、保护或淘汰认知帧，而由此得到的结构会直接成为后续认知表达式的求值上下文。因此，被学习的对象既是持久数据，也是后续计算状态的一部分。

本文用“经验迁移”或“测试时智能体学习”指称这种经由状态发生的过程；它不表示梯度更新、微调或基础模型权重变化。

现有长期记忆评测通常侧重从长历史中保持、检索和综合信息，LongMemEval-V2 是此类评测的代表 [22]。STATE-Bench 则检验智能体能否把先前任务形成的经验迁移到新的工具任务 [25]。本文关注的是认知帧形成后能否作为计算状态影响后续行动，而不只是从长历史中回答问题，因此 ME-07 采用 STATE-Bench。

对于领域 d，ME-07 中的机制可以概括为：

$$C_d^{(i+1)} = \mathrm{Commit}\Big(C_d^{(i)}, \Delta_\theta(\tau_i, C_d^{(i)})\Big),$$
$$a_q \sim \mathrm{Eval}_\theta\Big(e_q; \Pi_q(C_d^{(100)})\Big).$$

其中 τ_i 是一条已完成训练轨迹，Δ_θ 是模型提出的一组认知帧/关系变化，Commit 是经运行时校验的上下文事务，Π_q 是提供给测试任务认知求值的任务条件投影。经验问题不只是 τ_i 是否仍被存储，而是它经整合产生的影响是否保留在 $C_d^{(100)}$ 中，并参与 a_q 的生成。

ME-08 比较 Morphz 与传统智能体在复杂终端任务中的任务完成能力。为此，本文使用包含 89 个高难度命令行任务的 Terminal-Bench 2.1，以 Codex CLI 作为传统智能体对照，在相同模型和执行环境下进行配对评测 [23,26]。

2.5 大语言模型计算机与虚拟机

L2MAC 把大语言模型组织为存储程序计算机，用指令寄存表和文件存储完成长代码或长文本生成 [20]。Playbooks AI 将面向模型的汇编、编译器和运行时描述为人工智能系统的虚拟机或通用语言运行时 [21]。这些工作共同构成“大语言模型计算机”和“大语言模型虚拟机”的相关设计空间。

Morphz 的可检验主张是：**大语言模型在智能体自有的结构化状态上对递归认知符号进行非确定性求值；认知数据与候选程序共享项域；返回值可以进入后续计算；确定性运行时控制权威状态提交和现实副作用。**某些认知符号的语义依赖上下文，解释了它们为什么需要大语言模型；“非确定性认知求值器”则定义其一级计算身份。当前实现还允许模型返回新的可执行 Yao 程序，由运行时验证、持久化并显式执行。调度、持久化和恢复是这一架构的工程增强与派生能力。

3. 问题定义与设计目标

3.1 从消息历史到认知状态

设一个智能体在时刻 t 接收用户输入、工具结果、定时器或外部事件。传统对话式系统通常构造消息序列：

$$H_t = [m_1, m_2, \ldots, m_t]$$

并令模型生成下一条消息或工具调用。即使系统增加向量检索，返回值通常仍被拼接为文本片段。该表示可支持回忆，却不能自然保证对象身份、版本、作用域、因果和修订关系。

Morphz 将智能体的认知状态抽象为：

$$C_t = (F_t, R_t, O_t, B_t, V_t)$$

其中：

- F_t 是带身份、类型、内容、来源和生命周期的认知帧集；
- R_t 是认知帧/观察对象之间的派生、取代、冲突、归属等关系；
- O_t 是尚待消化或可追溯的观察对象；
- B_t 是当前表达式或局部过程中的绑定环境；
- V_t 是上下文的版本与事务元数据。

本文把这种可学习、可修订的持久认知状态单元称为**认知帧**。它不是模型内部不可见的激活，而是带来源、版本和关系边、可被投影到后续求值中的显式对象。

该定义不要求所有实现都使用相同字段，而要求至少满足：认知对象可被稳定引用，关系可显式表达，更新可追踪，结构可投影给模型，并能跨会话恢复。

3.2 设计目标

核心机制应满足以下六个目标：

1. **结构性。** 目标、事实、经验、计划、观察和关系不能只依赖段落位置区分。
2. **递归组合。** 复杂认知操作能够由较小项组合，且局部结果可被命名和引用。
3. **非确定性认知求值。** 大语言模型结合上下文和观察对象，对允许存在多个有效结果的认知表达式进行求值，而不是模拟唯一且由规则完全规定的状态转换。
4. **求值权分离。** 大语言模型对上下文依赖的认知符号进行非确定性求值，运行时负责预定义的确定性控制逻辑、权威状态提交和现实副作用；当前二者通过类型明确的程序输入与结果接口衔接。
5. **结果回流。** 求值结果不是终止文本，而能以兼容结构进入当前绑定或持久上下文；模型还可以在激活边界提出新的候选程序。
6. **状态与副作用隔离。** 非确定性认知值与权威状态提交及现实副作用之间存在验证、权限、事务和因果边界。

4. 认知符号求值模型

4.1 递归项语言

令 Σ 为认知算子与构造符集合，A 为字符串、数字、布尔值、标识符和引用等原子集合。表达式属于自由项代数：

$$e \in T_\Sigma(A) ::= a \mid (s\ e_1\ \ldots\ e_n)$$

其中 $a \in A$，$s \in \Sigma$。当前实现采用 S 表达式作为序列化形式，例如：

```
(resolve-conflict
  (experience (ref frame:old-policy))
  (evidence (ref observation:new-contract))
  (preference (ref frame:user-boundary)))
```

本文所说的“程序与数据采用同一递归表示”，是指表达式、作为值返回的结构和上下文中可计算的认知对象共享递归项表示，以及相同的构造、引用、遍历和变换机制。Lisp 的同像性已有长期研究，本文不将其本身作为创新点。

4.2 上下文投影

完整上下文可能大于模型窗口，且包含权限受限内容。运行时根据当前表达式、目标和策略生成投影：

$$\pi_t = \mathrm{project}(C_t, e_t, p_t)$$

其中 p_t 表示权限、预算和相关性策略。投影保留对象身份、关键关系与引用路径，而不是仅拼接无身份的文本摘要。模型可在需要时通过受控操作请求更多观察对象或认知帧。

4.3 非确定性认知求值器

本文把需要结合上下文进行语义判断、但仍受输入结构、算子说明、输出契约和验收条件约束的算子称为**非确定性认知符号**。它与 `+` 等语义由规则完全定义的确定性符号不同：程序员不必预先穷尽其全部判断规则，但它也不是不受约束的自然语言提示。

对固定的认知表达式、上下文投影、观察对象和求值内核，先定义满足可见上下文与输出契约的允许值集合：

$$\mathcal{A}(e_t, \pi_t, o_t, k) = \{v \in V \mid \mathrm{accept}(v; e_t, \pi_t, o_t, k) = 1\}.$$

当 $|\mathcal{A}(e_t, \pi_t, o_t, k)| > 1$ 时，该表达式在规范层面具有一对多的允许求值关系，本文沿用程序语义中“同一初始条件不唯一决定结果”的含义，把它称为非确定性认知求值 [27]。大语言模型以条件分布实现候选值的选择：

$$q_\theta(v \mid e_t, \pi_t, o_t, k), \qquad v_t = \mathrm{decode}(q_\theta)\,.$$

只有 $v_t \in \mathcal{A}(e_t, \pi_t, o_t, k)$ 时，该候选值才构成一次被接受的求值结果。因此，非确定性首先是规范层的一对多关系；概率分布与采样或贪心解码是模型选择候选值的实现机制。

其中：

- e_t 是待求值认知表达式；
- π_t 是上下文投影；
- o_t 是当前观察对象；
- k 是算子说明、示例和输出契约组成的求值内核；
- v_t 是文本、结构化数据、认知帧/关系候选、绑定值或候选程序项。

当前 `infer` 接受一个完整的 Yao 正文，而不是固定的任务与证据参数。该正文与 `eval` 使用同一套解析、类型检查和副作用分析；最外层算子决定由大语言模型还是运行时负责求值。在嵌套求值边界上，父程序只有在 Yao 源码的 `(captures NAME...)` 中显式列出某个绑定，运行时才会把该绑定的值随本次 `infer` 请求发送给当前配置的模型服务商；未列出的绑定和完整运行时环境不会被隐式附带。`captures` 只授权数据披露，不授予工具或对象操作权限。`infer` 的结果统一按声明的 Yao 类型解码，可以是 `String`、`Json`、命名/复合类型、引用类型或 `Program<T,E>`。若结果类型声明为 `Program<T,E>`（`T` 为输出类型，`E` 为允许的副作用集合），模型还可以直接返回一段以 `(eval ...)` 或 `(infer ...)` 为根的 Yao 候选程序；它必

须经过运行时解析、类型检查、求值所有者保留、有效副作用集合与权限上界检查、规范化、来源绑定和持久化，才能成为可执行程序值。对于 `infer` 根，从父程序进入新的正式子求值本身计入 `infer` 副作用。

在本文提出的计算模型中，大语言模型被定义为**非确定性认知求值器**。认知符号具有上下文依赖性：其求值结果不能仅由符号名称和参数唯一决定，还取决于当前结构化上下文和观察对象。认知算子仍然可以由输入结构、自然语言说明、输出契约和验收条件约束，但程序员不需要用传统代码预先穷尽其全部判断规则。

4.4 确定性运行时求值器

令 P_t 为经过验证和转换的计划，S_t 为权威运行时状态，x_t 为运行时已经观察到的外部输入，例如工具返回、时钟事件或调度事件。运行时求值是受规则约束的状态转换：

$$\mathrm{eval}_R : (P, S, X) \to (S', O, \kappa)$$

单步可写为：

$$\mathrm{eval}_R(P_t, S_t, x_t) \to (S_{t+1}, o_{t+1}, \kappa_t)$$

运行时对语义已经完全定义的算子求值，解析引用，检查权限和版本，调度现实动作，记录真实观察对象，并决定哪些状态转换能够提交。κ_t 表示完成、等待、失败或继续信息。本文所说的“确定性”只限定运行时在给定计划、已提交状态和显式外部输入后的验证、求值与提交决定；它不声称工具结果、并发调度、网络或外部环境本身是确定的。

4.5 认知求值与运行时求值的组合

Morphz 当前通过以下已经实现的路径连接非确定性认知求值和确定性运行时求值：

```
模型激活
   └─── 提出 eval 程序
         └─── 解析 + 校验 + 转换
               └── eval_R 执行预定义的确定性控制逻辑和现实操作

eval_R 正在执行有类型计划
   └─── 在 infer 节点挂起，并把完整 Yao 正文交给 infer_θ
         ├── 只有显式 captures 的父程序绑定随请求跨越边界
         └── infer_θ 求值该正文并返回声明类型的结果
               ├─── 数据值：运行时解码、校验、绑定并恢复原计划
               └── Program<T,E>：返回原始 Yao (eval ...) 或 (infer ...) 候选程序
                     └─── 运行时准入 + 按根节点求值所有者分派
                           ├── eval 根：持久子计划执行
                           ├── infer 根：正式子 Evaluation
                           └── 声明类型的结果汇合并恢复原计划
```

这比把任意模型工具调用直接当成权威操作具有更明确的结构。模型提出递归项，运行时拥有验证、引用解析、权限能力、调度和提交权；在已经验证的计划内，`infer` 把同一个完整 Yao 正文交给模型负责求值，并创建持久的子求值，其有类型结果成为确定性计划数据。

模型返回的程序只有在外层程序显式声明 `Program<T,E>` 结果、通过运行时准入并由 `(run PROGRAM)` 调用后才会执行。运行时重新验证当前权限能力，禁止捕获调用者局部绑定，并把已准入程序持久化为因果关联的子执行。`eval` 根由运行时按子计划推进；`infer` 根则立即建立正式子 Evaluation，并在取得有类型终值后恢复父计划。两类程序都可以继续包含或返回程序值，从而形成求值所有者之间的动态多次转移；程序嵌套、推理、工具调用和总子工作量受共享预算限制。现有确定性测试覆盖以 `eval` 和 `infer` 为根的程序值准入、有效副作用上界、持久子执行、求值所有者分派和重启后恢复，动态适应的真实任务效果则留待专门实验评估。

4.6 结构化结果回流

本文的核心闭环不是“生成一个结构化输出”本身，而是输出能够进入局部绑定，或经事务提交成为可被后续表达式引用的上下文对象。令规范化函数产生候选上下文变化和局部绑定：

$$n(v_t) \to (\Delta C_t, b_t)$$

其中 ΔC_t 是候选上下文变化，b_t 是可供局部表达式引用的绑定。经策略和事务边界提交后：

$$C_{t+1} = \mathrm{commit}(C_t, \Delta C_t, V_t)$$

后续求值使用：

$$\mathrm{infer}_\theta(e_{t+1}, \mathrm{project}(C_{t+1}), o_{t+1}, k)$$

当前系统支持三种结果回流：

1. **局部回流**： 工具或语义结果绑定为 `name.field`，在同一递归表达式的后续子树中引用；
2. **上下文回流**： 结果成为认知帧/关系，并在未来轮次或会话中被检索、修订或作为参数；
3. **程序值回流**： 嵌套 `infer` 可以返回以 `eval` 或 `infer` 为根的原始 Yao 候选程序；运行时准入后，显式 `run` 按求值所有者建立持久子执行，其结果再恢复父计划。

这一区分很重要。仅要求模型输出 JSON，或者将模型文本保存进数据库，并不自动构成认知符号回流；只有当结果保留可计算结构、身份和引用语义，并实际成为后续操作数时，闭环才成立。

4.7 权威状态提交与现实副作用边界

并非每个认知项都需要经过运行时，纯认知求值可以留在 infer_θ；语义已经完全定义的子树也可以连续由 eval_R 求值。不可改变的边界是：任何现实副作用或权威状态转换都必须经过确定性运行时。一个代表性的跨激活循环为：

$$\begin{aligned}(e_t, C_t, S_t) &\xrightarrow{\mathrm{infer}_\theta} (v_t, \Delta C_t^{\mathrm{cand}}, P_t^{\mathrm{cand}}) \\ &\xrightarrow{\text{validate+lower}} (\Delta C_t', P_t') \\ &\xrightarrow{\mathrm{eval}_R} (S_{t+1}, o_{t+1}, \kappa_t) \\ &\xrightarrow{\mathrm{project}} C_{t+1}.\end{aligned}$$

该分工同时保护两侧。把预定义的确定性控制逻辑和现实副作用交给 eval_R 不会削弱非确定性认知求值，而是保留大语言模型在其语义域内的自由；infer_θ 可以提出状态变化，但不能因此获得宣告现实已经改变的权限。若过早把上下文依赖的认知算子改写为固定规则，或把未经验证的认知候选直接提交为权威状态，都会破坏候选认知与已提交状态之间的边界。

5. Morphz 系统实现

5.1 分层结构

当前 Morphz 原型由四层构成：

1. **结构化上下文。** 保存上下文认知帧、关系、观察对象、版本及保护/退休状态，并向模型投影当前相关认知状态。
2. **认知项层。** 使用 S 表达式/Yao 表示可递归的认知数据、控制结构和候选程序。
3. **计划与验证层。** 检查含副作用的候选表达式，并将其转换为有类型计划中间表示。
4. **运行时层。** 通过调度器、持久事件记录、执行任务和交付机制执行、记录、恢复并路由现实结果。

这些层按权威而不是部署位置分工。非确定性认知求值覆盖上下文投影、认知符号求值和结构化候选生成；确定性运行时覆盖验证、对语义已完全定义的子树求值、调度与提交。模型可以构造认知值和候选程序，但只有运行时可以授权和记录现实状态转换。`eval` 与 `infer` 共享同一个经过解析和类型检查的 Yao 正文；更换最外层算子只改变求值所有者。嵌套 `infer` 也不会继承整个父作用域，只有源码通过 `captures` 显式列出的父程序绑定可以进入模型请求。

对话会话只负责输入输出路由，不拥有智能体的认知状态。多个会话可以挂载同一上下文；模型或运行时进程也可以更换，而智能体的结构化状态仍由持久存储恢复。这些能力是核心闭环的系统推论，但本文不把它们等同于核心机制本身。

5.2 Yao 语法树与统一语言卡

Yao 源码以 S 表达式作为表层语法，解析后的抽象语法树只包含原子节点和递归列表节点。语义不由这种简化树形本身决定：统一语言卡描述类型、值构造、控制结构和副作用算子，类型分析器再把源码转换为有类型中间表示。模型与运行时使用同一份语言卡和类型契约，而不是各自维护一套名称相同但含义可能不同的算子说明。

算子语义由语言卡、类型系统和运行时规范显式提供，而不完全依赖模型从 `seq`、`bind` 等名称中猜测。例如，规范规定 `seq` 的求值顺序、未被选择的 `if` 分支不得产生副作用、绑定的词法作用域、并行分支隔离，以及 `run` 只能执行已经准入的程序值。

例如：

```
(eval
  (requires (tools read))
  (seq
    (bind source
      (call read (path "docs/design.md")))
    (infer
      (captures source)
      (returns String)
      (seq
        (bind criterion "优先保留可由原文验证的主张")
        "依据 criterion 从 source 中提取核心假设和证据"))))
```

模型在记录的 Morphz 运行中曾自主构造与此同类的 `(eval (requires ...) (seq ... (infer ...)))` 程序以读取文件并总结。自主程序构造属于执行轨迹中的定性观察；ME-02 与 ME-05 使用固定的递归程序集合检验求值行为。

在该结构中，外层 `eval` 表示确定性运行时对预定义的控制逻辑与现实操作进行求值，内层 `infer` 表示由非确定性认知求值器执行其完整正文。父程序中的 `source` 只有因为被 `captures` 显式列出，才会随本次模型请求跨越求值边界。普通有类型结果以绑定值回到外层程序，而不是终止为一段与程序分离的自然语言回答；`Program<T,E>` 结果则以原始 Yao 源码返回，经运行时准入后由显式 `run` 按 `eval` 或 `infer` 根的求值所有者建立持久子执行，并在终止后恢复外层程序。

5.3 程序验证、有类型计划转换与副作用控制

候选 S 表达式不被直接当作现实命令执行。验证器先检查语法深度、调用数量、工具注册、权限声明、静态引用、类型、输出类型和副作用契约，再将表达式转换为有类型计划中间表示。当前有类型表示覆盖值构造、`Seq`、`Bind`、`If`、`Match`、`Fallback`、`Map`、`Infer`、`Call`、`Par`、`Run` 和 `Host` 等节点。`Infer` 保留完整的有类型 Yao 正文、显式捕获集合及结果类型；结果类型可以是 `String`、`Json`、命名/复合类型、引用类型或 `Program<T,E>`。实现还显式限制表达式深度、映射宽度、工具调用数、推理节点数、并行分支数、程序值嵌套和总子工作量。

有类型计划中间表示随后被桥接到调度器求值内核。该边界提供三项性质：

- 非确定性模型只能提出候选计划；
- 工具权限、预算和引用在执行前可检查；
- 现实观察对象与完成状态被写入持久事件记录，而不是只保留在对话文本中。

5.4 上下文事务与多版本并发控制

上下文更新通过显式事务完成。事务携带基础版本，可以派生、修订、保护、解除保护或退休认知帧，建立关系，并退休已被消化的观察对象。运行时对不相交修改支持自动变基，对重叠或陈旧写入返回冲突，从而避免并行的同级目标静默覆盖同一认知对象。

这一机制把“模型记住了什么”从不可检查的提示副作用变成可审计的状态变化。上下文认知帧的来源和版本可追踪，关系可表达派生与取代，活跃观察对象可在形成稳定认知帧后退休。结构化上下文因而同时承担工作记忆、长期认知和计算环境，而不仅是检索库。

当前运行时还对活跃用户请求的根观察对象设置围栏：与该请求关联的回复完成交付之前，上下文维护不能退休这条请求。请求因此在整个回复生命周期中保持为活跃操作数。

5.5 持久执行与继续信息

运行时为目标、求值、激活和交付保存持久状态。工具结果作为观察对象回到上下文，并通过继续信息触发后续求值。进程重启后，未完成的目标可以重新获得租约并继续执行；租约与围栏避免两个求值同时取得同一工作的执行权。该机制使“结构化结果回流”不依赖单次模型请求或单个进程内存。

6. 研究问题与实验

6.1 研究问题与证据边界

论文实验被组织为机制、系统与外部效度三层，回答六个问题：

- **RQ1——表示**：同一套递归结构及其求值语义分别以 S 表达式、JSON 抽象语法树或 Markdown 程序表示时，能否被求值而不产生表示特有的任务退化？
- **RQ2——认知状态**：结构化上下文能否跨会话、并发更新、进程重启和延迟使用保存并修订当前状态，同时相对消息或上下文压缩基线不降低最终决策正确性？
- **RQ3——非确定性求值**：一个认知符号能否拥有多个契约允许值，同时受可见上下文约束，并在不同模型家族上保持这一性质？

- **RQ4——提交与副作用边界**：确定性运行时能否阻止模型生成候选、陈旧事务、重放或不可信观察对象静默成为权威状态，或触发现实副作用?
- **RQ5——经验学习**：在读取相同已完成训练轨迹后，Morphz、完整开源智能体运行时 Letta 与 Memo 支撑的向量检索参照智能体，分别能以多高可靠性把程序性经验迁移到留出企业工具任务；并且在 Morphz 内部，由事务形成的认知帧是否确实构成留出求值所使用的上下文快照?
- **RQ6——外部效度**：在模型和环境相同的条件下，完整 Morphz 智能体能否在公开任务集上保持相对参考智能体的通用命令行任务能力?

本文区分正式批次原始分与事后诊断。事后语义重评分不修改模型输出，也不替代预先规定的严格分；当结构契约形状错误可以与语义值错误分离时，它回答另一个明确问题。若官方验证器超时或未返回结果，本文只在不调用模型、不修改冻结最终状态和官方测试的条件下顺序复评，并同时保留正式批次原始分。模型服务方拒绝和未通过官方验证器的任务计为失败；只有预先规定的完整性检查确认整次运行无效时，才排除该运行。基线不具备的架构能力记为“不适用”。

ME-01 ~ 03 是小样本受控机制实验；ME-04 检验权威状态提交与故障处理；ME-05 覆盖九种模型配置；ME-06 是使用 Morphz 完整状态链路的三个配对长程测试样例；ME-07 比较三个系统的经验迁移；ME-08 引入公开终端任务集。各实验回答不同研究问题，结果分别报告。

6.2 共同运行控制

模型机制实验统一通过 CLIProxyAPI 使用 GPT-5.6 Sol，采用最高推理强度、单候选且不允许回退到其他模型。各实验组使用独立状态目录、数据库、上下文与模型服务配置；执行二进制或运行器提交版本、确切模型标识、提示词、响应、用量和评分器产物均被记录。论文结果只纳入通过预先规定的实验完整性检查的运行。

ME-07 使用 STATE-Bench v0.8.1 [25]。三种系统在每个领域得到相同的 100 条规范训练轨迹、50 个未参与训练的测试任务（下文称“留出任务”，held-out tasks）和领域工具；被测智能体、用户模拟器与语言模型评审器均使用 GPT-5.6 Sol/max，并共享相同的模拟器提示词、评审器提示词和单次尝试规则。每个留出试次都从对应领域训练完成状态的隔离副本开始。内部提示词、状态表示、记忆管理、工具循环和调度属于被测系统差异。

STATE-Bench 的智能体学习评测不用字面记忆问答衡量记忆，而是检验历史经验能否迁移到留出工具任务。智能体在模拟购物、旅行或客服环境中对话，并可调用领域工具修改隔离的任务数据库。唯一主指标 `task_completion_pass@1` 只在两个二元条件同时通过时记 1：确定性 `state_requirements_met` 核对最终环境状态（包括不应修改时确实没有修改）；评审式 `task_requirements_met` 核对非状态的对话和程序要求，例如信息披露、用户确认、政策正确性和禁止性陈述。两个分项的平均值和 1 ~ 5 分用户体验作为诊断性次要指标单独报告。

ME-08 对 Terminal-Bench 2.1 全部 89 题进行完整双臂评测 [23,26]。Morphz 与 Codex CLI 0.149.1 对每题各运行一次，使用同一 GPT-5.6 Sol 最高推理强度、同一 Linux 主机、同一任务注册表摘要、实验组内并发 8 和零重试。论文采用最新修复版 Morphz 的完整运行与冻结 Codex 参考运行；二者遵循同一协议，但开始时间和模型服务账号不同。

ME-08 使用官方验证器的二元任务判定；正式运行缺失验证器结果时，按上一段规则对冻结最终状态进行零模型复评。在仅 Morphz 与仅 Codex 通过的任务上进行双侧精确二项检验（与该二元配对情形下的 McNemar 精确检验等价）。该检验量化按任务对齐的描述性证据；每题只有一次尝试，因而不能恢复题内采样方差。

6.3 ME-01～ME-06 核心结果

实验	设计	结果	可以支持的解释
ME-01	5 个任务族 × 追加式消息、结构化只读、完整 Morphz	15/15 严格成功	结构化上下文和结果回流没有使无容量压力任务退化；SQLite 持久化与事务路径确实执行
ME-02	6 任务 × S 表达式抽象语法树、JSON 抽象语法树、Markdown 程序	18/18 严格成功；各组 6/6	三种表示均完成全部任务，支持以同一递归结构表示程序和认知数据的可行性
ME-03	24 个基线/干预任务回合	非确定性条件 12/12 严格；上下文变化 6/6；确定性控制 11/12 严格、12/12 语义值正确	认知值受上下文约束且不必归约为唯一且完全预定义的状态转换；非确定性不要求采样多样性
ME-04	8 个权威状态提交/故障处理实验单元	8/8 通过；989 个库测试通过	候选认知与权威提交可以由运行时分权
ME-05	9 模型 × ME-02/03 核心子集	144/144 完整；98/144 严格；程序最终值 32/36；非确定性语义选择 104/108，且可解析响应 104/104	机制不依赖单一模型家族；精确结构契约与轨迹合规仍需要运行时处理
ME-06	3 长程测试样例 × 受控上下文压缩、完整 Morphz	两组均 3/3 语义成功、24/24 状态字段、3/3 唯一行动	Morphz 在保持最终结果的同时实际执行事务、恢复和审计路径

6.4 结构化上下文与结果回流

ME-01 比较完整消息历史、关闭直接上下文事务的 Morphz 结构化只读路径，以及完整 Morphz。五个任务族覆盖延迟引用、来源权威、显式取代、跨会话连续性与上下文隔离。15 个实验单元全部返回严格正确的最终行动。在没有容量压力时，完整消息历史足以解决这些任务；该实验建立的是结构化路径的非退化结果。

完整 Morphz 通过上下文事务工具提交认知帧，在后续激活中投影，终止进程后由 SQLite 恢复，同一上下文跨会话共享，并排除相邻上下文。前两组没有上下文事务能力，因此事务指标记为“不适用”。

ME-06 把问题推进到三个包含 120 个事件、12 个检查点的长程测试样例。受控上下文压缩参照在检查点 6 前固定进行一次上下文压缩；Morphz 使用实验中配置的 262,144 词元输入容量，不制造人工压力。两组均得到 3/3 测试样例语义成功、24/24 最终状态字段和 3/3 唯一行动。Morphz 额外执行 40 次成功认知帧事务；原始轨迹记录跨会话连续性、版本冲突后的重读、进程重启恢复、上下文隔离和完整因果审计。

6.5 等信息表示对照

ME-02 从同一个有类型规范程序中间表示生成三种渲染形式。六个任务覆盖嵌套顺序、分支、绑定、共享引用、回退与交替分支。S 表达式抽象语法树、JSON 抽象语法树和 Markdown 程序均为 6/6，通过次数、平均模型请求和工具调用完全相同。

由于三组均为 6/6，这组任务存在天花板效应，不能据此判断一种表示优于另外两种。三种表示共同说明，本文贡献在于程序、认知数据和返回值采用可寻址、可求值的递归结构，而不在于某一种表层语法。S 表达式是当前紧凑、同像且易于校验的实现。

6.6 受上下文约束的非确定性求值与跨模型普适性

ME-03 设计三类任务，每个认知表达式有 3～6 个契约允许值。可见上下文选择其中一个合法子集；干预修改相关认知帧，使前后合法集合不相交。12 个非确定性任务回合全部满足严格结构契约与可见契约，六个配对干预全部把选择移动到干预后的合法集合。两次重复往往选择相同值，这与定义一致：非确定性是一对多的允许求值关系，不要求重复采样表现出高熵。

确定性控制只允许唯一值，严格分为 11/12；剩余响应选值正确，但把数组字段序列化为字符串。ME-05 分别报告严格结构合规与语义正确性。

ME-05 在 GPT-5.6 Sol、Claude Opus 5、Grok 4.6、Gemini 3.7 Flash High、DeepSeek V4 Pro/Flash、Kimi K3-256K、GLM-5.3 和 Qwen 3.8 Max Preview 上运行 ME-02/03 的同一核心子集。144 个实验单元全部结束，严格汇总为 98/144。程序最终值为 32/36；四个未交付均为 Claude 模型服务方的安全策略拒绝。非确定性求值的预先规定严格分为 72/108。明确标注的事后诊断忽略 `basis` 数组/字符串和额外字段等形状差异，再把 `selected` 放回原始可见契约校验，得到 104/108；所有 104 个可解析选择均合法。该结果支持跨模型语义可行性，同时说明确定性解析、校验、转换与失败处理是必要边界。

6.7 运行时对权威状态与现实副作用的控制

ME-04 不调用模型。八个实验单元使用 Morphz 运行时组件和确定性故障注入，覆盖工具准入、伪造/非法程序值、篡改重放状态、认知帧多版本并发控制、相交/不相交并发更新、重复事件/事务身份、幂等与非幂等副作用恢复、上下文事务已提交后的继续信息、恶意观察对象，以及主体/会话/上下文隔离。八项全部通过，完整库测试套件为 989 项通过。

结果表明，可见文本无法扩大运行时当前开放的工具边界；实验覆盖的陈旧写入、重放和崩溃窗口均按显式语义被拒绝或恢复。

6.8 经验学习智能体系统对照：STATE-Bench

ME-07 比较三个端到端系统边界：使用结构化上下文与上下文事务的 Morphz；使用原生核心记忆、召回记忆与归档记忆的完整开源智能体运行时 Letta 0.16.8（其架构源自 MemGPT [15]）；以及采用 Memo 2.0.19 写入时学习和 `search(top_k=3)` 的参照智能体 [24]。Memo 实验组使用本地 Qdrant、`nomic-embed-text` 向量模型和基础向量检索配置，本文称其为 Memo 支撑的向量检索参照系统。

ME-07 报告两层相互区分的证据。主任务得分比较三个完整系统边界的端到端效果；Morphz 的只读轨迹审计检查机制是否跨越训练与测试边界：每个领域训练会话把全部 100 条训练任务回合通过上下文事务提交；训练后的投影包含带训练来源的活跃认知帧与显式关系；隔离留出会话中的每次模型求值绑定到该领域的最终训练版本。审计表明，经验已被整合为结构化认知状态，并实际参与后续求值。

实验包含三个领域及每领域全部 50 个留出任务，每题在三个系统上各运行一次，即每组 150 次、总计 450 次。一个配对实验单元是同一领域的同一任务；最多并发四个实验单元，实验单元内三组并行，每次运行使用隔离的训练后状态副本。每个终态结果都原子写入；计分失败保持为零，且不静默重试。主要比较为 Morphz–Letta 和 Morphz–Memo，按任务聚类计算区间、执行符号翻转检验并作 Holm 校正；次指标包括通过@1、要求满足、用户体验、词元、墙钟、训练成本与失败分类。

全部 450 次运行均形成终态产物并通过预先规定的实验完整性检查。Morphz 通过 122/150（81.33%），Letta 为 93/150（62.00%），Memo 支撑的参照系统为 96/150（64.00%）。状态要求通过率分别为 92.67%、79.33%、82.67%；任务要求通过率为 85.33%、68.00%、70.00%；平均用户体验为 4.410、3.625、3.833。Morphz 1 个、Letta 7 个、Memo 4 个终态失败均保留并计零。

系统	完整通过	状态要求	任务要求	平均用户体验	终态失败
Morphz	122/150（81.33%）	92.67%	85.33%	4.410	1
Letta	93/150（62.00%）	79.33%	68.00%	3.625	7
Memo 支撑的参照系统	96/150（64.00%）	82.67%	70.00%	3.833	4

Morphz 相对 Letta 高 19.33 个百分点（按任务聚类的自助法 95% 置信区间 [+10.67,+28.00]），相对 Memo 高 17.33 个百分点（[+10.00,+24.67]）。原始配对符号翻转检验 `p` 分别为 0.000040 和 0.000030，Holm 校正后均为 0.000060。这些效果量比较的是完整端到端系统边界：协议固定训练轨

迹、留出任务、基础模型、领域工具、评测器和单次尝试规则，但有意把各系统内部提示词、记忆表示、调度器和工具循环作为被测系统的一部分。

Morphz 只读审计在 150 条留出轨迹上全部通过。每个领域的训练均以 100 次已提交上下文事务结束于第 100 版；三个训练后投影合计包含 144 个活跃认知帧、395 条活跃关系与 795 个已退休对象。每个留出任务的首次模型调用都使用对应的第 100 版上下文；后续版本和事务哈希形成非递减、连续链。其中三个购物领域留出任务又执行了六次上下文事务。训练经验形成的结构化认知状态实际参与了全部留出求值。

6.9 外部终端任务评测：Terminal-Bench 2.1

ME-08 采用 Terminal-Bench 官方验证器的二元任务判定。论文比较最新修复版 Morphz 与冻结 Codex 参考运行；两者使用相同的 GPT-5.6 Sol 最高推理强度和模型服务线路，在同一 Linux/amd64 主机、同一任务版本、完全授权、实验组内并发 8 和零重试条件下，对全部 89 题各运行一次。两次完整运行采用同一冻结协议，但并非同时启动。

智能体	通过	正确率	**Wilson 95%** 区间
Morphz	74/89	83.15%	[74.04%, 89.51%]
Codex CLI 0.149.1	74/89	83.15%	[74.04%, 89.51%]

逐题对齐后，仅 Morphz 通过 7 题、仅 Codex 通过 7 题、共同通过 67 题、共同失败 8 题。Morphz−Codex 的差值为 0 个百分点；按任务固定种子重采样的 95% 区间为 [−7.87,+7.87] 个百分点，双侧精确配对检验 `p=1.0`。

两次完整运行还提供了系统级工程对照。Morphz 的模型服务方报告输入加输出为 67,387,261 词元，Codex 为 83,361,987；按 89 个已尝试任务分别为 757,160 与 936,652 词元，Morphz 少 19.2%。Morphz 端到端墙钟为 6,349.93 秒（1.76 小时），Codex 为 7,065.16 秒（1.96 小时），前者短 10.1%。两者的输入缓存率分别为 29.88% 和 93.12%。Morphz 运行与缓存传输实验组同时执行，因此墙钟只作为描述性工程观察。

请求轨迹显示，缓存率差异主要来自上下文的传输方式。该版本 Morphz 默认把完整结构化上下文编码为一条消息，并在每轮请求中重新生成；本次使用的 OpenAI Codex 订阅线路无法稳定复用这条变化消息中保持不变的内部前缀。Codex 以多条消息追加历史，已发送的消息保持不变。为此，本文单独编译了一种尚未进入默认运行时的实验性传输策略，在模型服务边界把结构化上下文分为稳定内容和追加式增量。该策略只改变请求编码，不改变状态模型；当模型服务能够复用单条消息内部的前缀或支持相应缓存断点时，无需采用这种适配。

为检验该策略，本文在完整 89 题上进行同期 A/B。两组同时启动，使用同一 Morphz 运行时版本、GPT-5.6 Sol 最高推理强度、模型服务账号和完全授权；每题运行一次，不重试，组内并发均为 8。对照组每轮重新发送完整结构化上下文，实验组使用追加式增量传输。正式批次原始得分为 72/89 和 71/89。四项任务的验证器未能在包含环境冷启动的 900 秒上限内返回结果；保持最终文件和官方测试不变且不再调用模型的顺序复评得到三项通过、一项失败。实验组另有一项在截止前约 57 秒形成最终正确文件，但终态回复和验证器随后超时；同一文件复评通过，故计入任务正确数，同时保留执行超时记录。复评后两组分别为 74/89 和 73/89；共同通过 68 题、共同失败 10 题，仅对照组通过 6 题，仅实验组通过 5 题，实验组减对照组为 −1.12 个百分点，双侧精确 McNemar `p=1.0`。两组共 178 次任务运行均通过预先规定的持久化、执行轨迹和结果完整性检查，且未使用额外的评测专用提示。

传输方式	复评后通过数	输入缓存率	未缓存输入词元	输入加输出词元	墙钟
每轮发送完整上下文	74/89	29.88%	46,285,487	67,387,261	6,349.93 秒
实验性追加增量传输	73/89	86.27%	8,669,562	64,979,430	6,765.76 秒

实验性追加式编码使输入缓存率提高 56.39 个百分点，未缓存输入减少 81.27%。模型服务方报告的输入加输出仅减少 3.57%，实验组输出词元增加 32.67%、推理词元增加 44.09%，墙钟增加 6.55%。因此，这项传输实验显著改善了缓存复用，但没有在本次运行中缩短端到端时间。

同期 A/B 同时运行两个并发 8 的实验组。配备 16 个逻辑处理器和 61.52 GiB 内存的主机平均已用内存为 5.14 GiB、峰值为 9.62 GiB，1 分钟平均负载为 5.439、峰值为 25.648。采样覆盖整台主机和两个实验组，不用于估计单个智能体的资源占用。

两次同协议完整运行均为 74/89，但差值区间较宽且跨越零，因而不能据此建立优越、等价或非劣结论；统计效力与区间边界见第 8.4 节。

7. 结果解释

7.1 已实现机制在经验上可达

组合证据支持论文的核心机制主张：模型能够求值递归结构化项；一个认知表达式可以拥有多个上下文合法值，并在相关上下文干预后改变；真实工具值和认知值可以成为绑定或持久认知帧；确定性运行时能把候选认知与权威状态/副作用分开。这些路径不局限于一个模型家族，尽管精确结构契约和执行轨迹合规在模型间明显不同。

结构化上下文还把追加式消息列表不直接提供的能力变成运行时原语：稳定认知对象身份、显式来源与取代、版本事务、跨会话状态、独立于进程的恢复、上下文隔离和因果审计。ME-01 与 ME-06 表明，在这些配对任务中，获得上述能力没有降低最终正确性。ME-07 提供了更大规模的留出任务证据：Morphz 在 150 个匹配任务上高于两个公开对照系统，轨迹审计同时确认每个 Morphz 任务都从训练后的认知帧投影开始。

7.2 机制证据与系统级证据

ME-01 ~ ME-06 分别隔离表示、状态回流、非确定性求值、权威边界、跨模型执行和长程状态处理。ME-07 比较三个系统的经验迁移，并以轨迹审计检查 Morphz 认知帧链路；ME-08 比较 Morphz 与 Codex CLI 的终端任务能力。这样的分层设计把计算机制连接到端到端行为，同时让不同研究问题分别由对应证据回答。

当前 `infer` 接口已经能够把普通有类型值或一等 `Program<T,E>` 结果返回确定性计划。以 `eval` 或 `infer` 为根的程序值准入、按求值所有者分派的持久子执行与恢复已经实现；尚未完成的是它在真实动态任务中相对固定脚本或普通重新规划的效果评测。

7.3 如何使用公开任务结果

ME-07 与 ME-08 回答互补的外部效度问题。ME-07 测试经验能否迁移到未见的有状态企业工具任务，其轨迹审计把端到端得分连接到实际训练过的结构化上下文；Terminal-Bench 在共享模型与环境下测试 Morphz 与 Codex CLI 的终端任务能力。ME-01 ~ ME-06 提供把这些系统结果连接到计算模型的受控机制证据。

综合已完成实验，Morphz 实现了持久、可事务的认知状态，并在多个模型家族上展示受上下文约束的非确定性认知求值。ME-07 中，Morphz 的留出任务通过率为 81.33%，Letta 为 62.00%，Mem0 支撑的参照系统为 64.00%；150/150 轨迹审计确认训练后的认知帧状态参与每个 Morphz 留出任务。在同模型、同主机和同冻结协议的 Terminal-Bench 完整运行中，最新修复版 Morphz 与 Codex CLI 0.149.1 均为 74/89；此次单次采样没有解析出稳定的系统差异。

8. 局限与有效性威胁

8.1 构念有效性

ME-01 ~ 03 隔离的是范围较窄的机制性质，任务规模和难度不足以估计稳定效应量；其结果说明机制可执行且没有观察到明显退化。ME-04 测试确定性边界而不是模型智能。ME-06 把长期状态、并发与恢复组合起来，但只有三个项目形态测试样例。没有一个总分能够完整表示"持久认知"，因此本文分开报告状态正确性、行动正确性、契约有效性、运行时事件和外部任务得分。

结构化表示可能提高可审计性而不提高答案准确率；系统级基准评测的差异也可能来自提示词、工具策略、收敛行为或模型服务方稳定性，而不是上下文。ME-07 通过执行轨迹审计减少了这种歧义，但没有在保持其他智能体内部完全不变时随机化记忆机制。因此，本文把它描述为系统级经验迁移结果，不把 ME-07 或 Terminal-Bench 的全部差异单独因果归于结构化上下文。

8.2 内部有效性

所有报告结果都绑定精确协议、二进制哈希、运行器提交版本和评分器版本。只有预先规定的完整性检查确认实验装置无效时，才排除对应运行；原始产物继续保留在审计材料中。

本文报告两个事后语义诊断。ME-05 把结构契约形状错误与合法认知选择分开；ME-06 接受语义等价的决策规则，以及与来源别名等价的具体证据事件标识。两者都不修改响应、不替代执行前定义的严格评分，且对所有实验组使用同一确定性规则；其证据强度仍低于执行前定义的语义评分。

ME-08 另对缺失官方验证器结果的任务进行零模型顺序复评。复评保持冻结最终文件、任务摘要和官方测试不变，正式批次原始分仍在审计材料中单独保留；Codex 的超时任务也按同一规则复核。

8.3 外部有效性

ME-01 ~ 03 使用合成任务，ME-05 虽扩大到九种模型配置，仍使用同一小型任务核心。ME-06 只有三个配对长程测试样例，不能据此推断广泛的记忆泛化。ME-07 覆盖 STATE-Bench 的三个模拟企业领域、150 个留出任务和两个公开对照系统；三种系统均使用 GPT-5.6 Sol/max，每题运行一次。ME-08 使用完整公开任务和 Codex CLI 对照，每题同样只运行一次，而且两组共享同一 CLIProxyAPI 路由及其策略限制。

实验尚未覆盖真实跨月组织运营、机器人、跨用户隐私或错误信念长期累积，也没有证明同一状态模型适合所有智能体类型。

8.4 统计效力

ME-01 有五个配对任务族，ME-02 每种表示六题，ME-03 有六个配对干预，ME-06 只有三个配对测试样例。这些小样本受控实验不足以估计统计显著的系统差异。ME-05 报告跨模型计数，但每个模型没有重复采样方差。ME-07 有 150 个配对留出任务，报告按任务聚类的自助法区间、配对符号翻转检验和 Holm 校正；区间不跨零，但每题一次仍不能估计题内采样方差，其评审式任务要求与用户体验也尚未经过独立人工校准。ME-08 包含 89 个按任务对齐的样本，双侧精确 `p=1.0`，任务级自助法差值区间为 [−7.87,+7.87] 个百分点；同样因每题一次不能估计题内采样方差，且区间过宽，不足以作等价或非劣结论。后续实验性缓存传输 A/B 也包含 89 个配对任务；验证器复评后，实验组与对照组相差 −1.12 个百分点，双侧精确 McNemar `p=1.0`，该单次配对同样不能证明两种传输方式等价。

8.5 效率与测量有效性

模型服务方的词元字段和推理词元语义在协议间不同，因此完整智能体效率比较使用同模型、同任务、同主机和同冻结协议的 ME-08 完整运行。Morphz 的报告总逻辑词元少 19.2%，端到端墙钟短 10.1%；由于 Morphz 运行与缓存传输实验组同时执行，墙钟只作描述性观察。

后续 89 题同期 A/B 单独启用实验性缓存传输：输入缓存率由 29.88% 提高到 86.27%，未缓存输入减少 81.27%；验证器复评后的通过数为 74 和 73，配对 `p=1.0`。实验组的请求数更少，但输出词元增加 32.67%、推理词元增加 44.09%，墙钟增加 6.55%。这些指标分别描述缓存复用、模型生成量和端到端执行时间，不能互相替代。

8.6 系统成熟度与术语

Morphz 运行时仍在演进。宽上下文投影、重复维护、认知帧增长、错误整合、所有权冲突和部分对话已提交但未交付的恢复仍需工程工作。由确定性运行时控制权威状态提交，可以减少一部分失败，但不能让模型生成认知天然正确或安全。

“非确定性认知求值”指受上下文条件约束的一对多允许关系。

9. 后续工作

9.1 扩大机制实验规模

更大规模的机制实验应使用未见测试样例，使旧证据、来源权威、延迟行动、竞争更新和持续上下文压力同时作用，并与强现代上下文压缩基线比较。还可以在 Morphz 内部单独改变认知帧投影或整合策略，同时保持其余运行时不变，以检验这些机制的因果贡献。ME-07 的评审式指标还需通过盲化人工抽样校准；重复尝试则可用于估计题内方差。

9.2 端到端效率评测与运行时优化

后续效率评测应继续使用匹配的终端任务工作负载，分别改变投影粒度、认知帧拆分、工具结构契约大小、维护触发策略和无变化状态抑制；同时报告总输入、未缓存等效输入、缓存输入、输出、验证成功率、维护调用、墙钟时间，并在使用真实接口计费时报告费用。目标不是为压缩而压缩，而是在保留稳定身份、来源与事务语义的前提下改善经验证的端到端权衡。

9.3 程序值动态适应实验

当前实现已允许 `infer` 在观察环境后返回一段以 `eval` 或 `infer` 为根的原始 Yao 候选程序，并由运行时完成解析、输出类型/有效副作用集合/权限校验、求值所有者保留、规范化、来源绑定、持久子执行和恢复汇合。后续实验将把该机制与固定脚本、普通重新规划和只生成一次计划的路径比较，评估模型生成程序的可靠性，以及程序值回流对后续行为的实际影响。现有测试已经验证接口、隔离和恢复，真实任务中的动态适应效果仍需实验测量。

9.4 学习求值器与上下文策略

ME-05 表明语义可行性较广，而精确契约合规随模型变化。后续应区分：上下文内算子学习、监督式结构生成、事务时机偏好优化，以及派生、修订、保护、整合和退休策略。训练必须保留运行时权威边界，而不是让模型在文本中假装现实已经提交。

9.5 长期部署与公开产物

更长实验需要按周观察错误信念积累、来源衰减、跨用户信息流、模型迁移、多会话协调和部分副作用恢复。公开产物应包含实验所用的精确提示词、测试样例、不含密钥的模型与服务配置清单、评分器源码、不可变原始结果、校验和、统计脚本，以及精确运行时二进制或可复现构建身份。

10. 讨论

10.1 对“智能体”的重新界定

在 Morphz 中，智能体不是一次模型调用，也不是一个会话，更不是某组模型权重。智能体的连续性来自可恢复的结构化上下文、持久事件和未完成继续信息；模型是可替换的认知语义求值器，会话是输入输出通道，运行时是现实状态与副作用的执行者。

该定义解释了为什么结构化上下文是其他产品特性的基础：没有可共享和可版本化的认知状态，多会话只能共享文本历史；没有可引用的认知帧与局部结构，线程并发难以表达共同认知和修改冲突；没有主动可操作的上下文，所谓“自维护”只能由外部摘要器代劳。

10.2 可学习的认知符号求值器

传统 CPU 和虚拟机按照规范执行预定义的确定性语义。Morphz 保留对这类操作的确定性处理，同时用大语言模型求值那些其值分布在算子求值内核、上下文、观察对象、示例和模型参数中的认知符号。认知求值器可以通过提示词、示例、微调或偏好优化得到校准，而权威状态转换和现实副作用继续由显式规则控制。

一个可行方向是把算子划分为三类：

- 语义由规则完全定义、必须由运行时精确执行的确定性算子；
- 可由模型求值且具有结构化验收条件的有界非确定性算子；
- 上下文依赖的认知算子，主要依据当前上下文和世界知识进行求值。

这种划分比“所有东西都交给大语言模型”或“把所有语义都写成代码”更能保留二者各自优势。当前有类型的 `eval/infer` 连接使部分边界成为可执行机制，但本文的主要研究对象是大语言模型在结构化状态上对认知项进行非确定性求值。

10.3 有界程序值回流与动态运行时适应

当前运行时已经支持一等 `Program<T,E>` 结果。`infer` 可以在观察环境后直接返回一段以 `(eval ...)` 或 `(infer ...)` 为根的 Yao 候选程序；运行时不会执行这段原始源码，而是先完成解析、输出类型/有效副作用集合/权限校验、求值所有者保留、规范化、哈希、来源绑定和持久化。显式 `run` 随后创建持久子执行：`eval` 根按运行时子计划推进，`infer` 根立即分派为正式子 Evaluation；两者终态都以声明类型接回父计划。

这一机制使两个求值器能够发生有界的动态多次转移，同时把权限能力、程序嵌套、推理、工具调用和总工作量限制在运行时预算内。它为机器人和长期自动化中的动态行为适应提供了可执行基础，但当前证据只验证了实现、隔离和恢复边界；模型在真实任务中生成程序的可靠性、相对固定脚本或普通重新规划的收益，以及外部非幂等副作用下的端到端性质仍需单独实验。

10.4 为什么采用 S 表达式，以及它未必胜出

S 表达式提供小型语法、显式树结构、直接递归组合以及代码即数据的自然表示。根算子容易识别，绑定、作用域、顺序、分支和认知帧结构也不需要为每类表达式另建语法。

JSON 拥有更成熟的生态和模式工具，自然语言更贴近预训练分布，有类型抽象语法树可能提供更强校验。Morphz 的计算模型不依赖 S 表达式这一表层语法；若其他表示在等信息实验中更可靠或高效，可以保留非确定性认知求值、结构化上下文、结果回流和确定性提交，仅替换表层表示。

10.5 安全与可审计性

结构化表示不自动等于安全，但它使安全边界可见。模型生成的候选表达式可以被类型检查，认知帧更新可以被版本化，工具调用可以被权限门拦截，观察对象可以绑定因果来源，最终副作用可以形成可审计的权威状态记录。未来还需要引入信息流标签、来源可信度、污点传播和人类审批，尤其是在上下文中存在不可信网页或跨用户数据时。

11. 结论

本文提出 Morphz 的核心计算假设：大语言模型可以在结构化上下文上对递归认知符号进行非确定性求值。认知帧、关系、观察对象、绑定、来源和版本构成独立于单次对话与模型调用的持续认知状态；程序和认知数据属于同一递归项域；结构化结果可以成为局部绑定或持久认知帧/关系。确定性运行时验证候选程序和上下文变化，并保留对权威状态提交和现实副作用的控制权。S 表达式是这一思想的当前实现载体，而不是主张本身；认知符号的上下文依赖性说明为什么需要语义求值，而“非确定性认知求值器”才是本文赋予大语言模型的计算身份。

当前实现和 ME-01 ~ ME-06 结果说明，这一方向已经形成可执行系统。模型在三种表面表示中消费结构化程序，真实观察对象重新进入局部数据流，上下文干预改变认知符号的允许值，九种模型配置完成同一机制矩阵；确定性运行时通过八项权威与故障测试。长程受控实验中，Morphz 在所有最终状态字段和行动上匹配受控上下文压缩参照，同时真实执行事务、多会话、版本冲突恢复、重启、隔离和审计。ME-07 又把证据从机制执行推进到经验迁移：Morphz 在 150 个留出有状态任务中通过 81.33%，Letta 为 62.00%，Memo 支撑的参照系统为 64.00%；150/150 轨迹审计确认每个 Morphz 任务都从对应领域训练后的第 100 版结构化上下文开始。

已完成实验表明，Morphz 实现了持久、可寻址、可事务的认知状态。在 ME-07 中，其留出任务通过率为 81.33%，Letta 为 62.00%，Memo 支撑的参照系统为 64.00%。在同模型、同主机和同冻结协议的 Terminal-Bench 完整运行中，最新修复版 Morphz 与 Codex CLI 0.149.1 均为 74/89（83.15%）；按任务对齐的差值为 0 个百分点，95% 自助法区间为 [−7.87,+7.87]。Morphz 的总逻辑词元少 19.2%，本次端到端墙钟短 10.1%。后续完整工作负载 A/B 表明，实验性传输策略可把输入缓存率从 29.88% 提高到 86.27%；验证器复评后，两种传输方式的通过数为 74/89 和 73/89。相关结果把认知状态机制连接到受控行为、经验迁移与终端任务执行；其有效性边界集中列于第 8 节。

有界程序值回流已经实现，为动态运行时适应提供了可执行路径；其在真实动态任务中的可靠性与收益仍待实验验证。本文的中心贡献，是把非确定性认知符号求值、智能体自有结构化计算状态、结构化结果回流，以及由确定性运行时控制权威状态提交和现实副作用的机制组合成一个可运行系统。

附录 A：端到端示例

下面给出一个示意性闭环。第一步读取现实证据，第二步由大语言模型进行非确定性认知求值，第三步把结构化结果作为认知帧修订提交，第四步在后续任务中直接引用该认知帧：

```
(eval
  (seq
    (bind contract
      (call read_file (path "contract-v2.md")))

    (bind decision
      (infer
        (captures contract)
        (returns String)
        (seq
          (bind criterion
            "比较新合同证据、既有交付政策和用户风险边界")
          "依据 criterion 和 contract，给出修订后的交付政策及理由")))

    (call context_tx
      (base-version (ref context.version))
      (revise
        (frame-id "delivery-policy")
        (content decision)
        (derived-from contract.observation-id)))

    (reply
      (explain (ref frame:delivery-policy)))))
```

该例中 `decision` 不是只用于生成当前回复。它被写入具有稳定 ID、来源关系和版本的认知帧；下一会话、另一目标或另一个模型可以把 `frame:delivery-policy` 当作结构化操作数继续求值。

附录 B：主张—证据矩阵

主张	证据状态	当前证据	尚缺证据
结构化上下文和持久结果回流在无容量压力的简单任务中不造成可见退化	受控机制实验	ME-01：5 个任务族、3 个实验组，共 15/15 严格成功；完整 Morphz 组真实使用 SQLite、上下文投影、`context_tx`、跨会话访问、进程重启和上下文隔离	更大留出任务，以及预先规定非劣界值的实验设计
程序和认知数据共用同一递归表示的机制不依赖 S 表达式表层语法	受控机制实验	ME-02：同一规范中间表示下，S 表达式抽象语法树、JSON 抽象语法树和 Markdown 程序各 6/6	更复杂程序，以及按表示分别测量的可靠性和效率
非确定性认知符号的允许值受上下文约束，并可包含多个合法结果	受控机制实验	ME-03：非确定性条件 12/12；6/6 配对上下文干预改变允许结果；确定性对照严格 11/12、语义值 12/12	更多未见算子族，以及执行前定义的语义评分
模型生成候选认知时，权威状态提交和现实副作用仍由确定性运行时控制	确定性验证	ME-04：8/8 提交与故障实验单元通过；权限能力、版本、重放、副作用和恢复边界均有专门测试	外部对抗安全评测和端到端形式化性质
机制可跨模型家族执行，但精确结构合同可靠性随模型而变	跨模型受控实验	ME-05：九种模型的 144/144 个实验单元完成；严格 98/144；程序最终值 32/36；非确定性求值语义诊断 104/108；另有 4 个模型服务方拒绝	重复采样、更大任务核和独立模型服务部署
长程结构化状态在增加事务与恢复语义时可以保持最终结果	长程受控实验	ME-06：受控上下文压缩与完整 Morphz 均为 3/3 测试样例、24/24 最终状态字段和 3/3 行动；Morphz 真实执行 40 次上下文事务，以及冲突重读、重启、隔离和审计	更多测试样例、持续上下文压力和更强统计设计

主张	证据状态	当前证据	尚缺证据
训练后的结构化认知状态能够把经验迁移到留出有状态任务	系统级外部对照与机制轨迹	ME-07：Morphz 122/150（81.33%）、Letta 93/150（62.00%）、Memo 支撑的参照系统 96/150（64.00%）；全部 150 个 Morphz 任务均从经审计的对应领域第 100 版训练上下文开始，三个领域的训练后上下文合计有 144 个活跃认知帧和 395 条关系	盲化人工校准、题内重复采样，以及 Morphz 内部整合/投影因果消融
在同模型、同任务、同主机和同冻结协议的 Terminal-Bench 完整运行中，Morphz 与 Codex 的单次分差尚未被解析为稳定系统差异	单次外部对照	ME-08：最新修复版 Morphz 与 Codex CLI 0.149.1 均为 74/89；差 0 个百分点，任务级自助法 95% 区间 [−7.87,+7.87]，精确配对 `p=1.0`	重复尝试以估计题内方差；若要主张等价或非劣，还需预先规定判定界值
在同协议 Terminal-Bench 完整运行中，Morphz 使用较少的报告总逻辑词元和较短的端到端时间	描述性观察	ME-08：Morphz 6,738.73 万词元，Codex 8,336.20 万，前者少 19.2%；端到端墙钟分别为 6,349.93 秒和 7,065.16 秒，前者短 10.1%	重复匹配试验与分机制消融
实验性追加增量传输能够提高完整任务集上的输入缓存复用	完整工作负载同期 A/B 与验证器复评	89 题对照组与实验组复评后分别为 74/89 和 73/89，配对 `p=1.0`；输入缓存率 29.88%→86.27%，未缓存输入减少 81.27%；墙钟增加 6.55%	重复尝试、更多服务线路，以及是否有必要将该适配纳入默认路径的评估
同一个完整有类型 Yao 正文可以由 `eval` 或 `infer` 求值，并把结果恢复到父计划	已实现并观察	求值所有者对称的正文解析与类型检查、源码级 `captures` 披露边界、持久子求值、声明类型统一解码、绑定测试及真实模型 Smoke	更广的认知工作流与故障注入重复
`infer` 结果能够在不重新部署时成为新的可执行程序并接回原计划	确定性验证	原始 Yao 程序结果、以 `eval` 或 `infer` 为根的 `Program<T,E>` 准入、输出类型/有效副作用集合/权限上界、求值所有者分派、内容哈希、调用者局部变量隔离、持久子执行、共享嵌套预算和重启恢复测试	真实模型动态适应任务、重复样本和外部副作用端到端实验

附录 C：复现入口

下面列出源码级入口。ME-07 与 ME-08 分别对应评测仓库的精确提交版本；完整复现实验还需要使用正文与产物清单记录的协议、模型与服务配置、运行时二进制、原始结果和评分器版本。

```
# ME-01、ME-02、ME-03、ME-06 机制实验
cargo run -q -p morphz-evals --bin me01_context_reentry_eval -- --help
cargo run -q -p morphz-evals --bin me02_representation_eval -- --help
cargo run -q -p morphz-evals --bin me03_bounded_open_eval -- --help
cargo run -q -p morphz-evals --bin me06_long_horizon_eval -- --help

# ME-04 权威状态提交与故障处理测试；ME-05 跨模型诊断
cargo test -p morphz --lib
python3 morphz-evals/tools/analyze_me05_results.py --help

# ME-07：评测源码提交 e32d3e165d35a69451d73845470641dfd02ba156
python3 -m benchmarks.state_bench.v2.run_public_systems_formal --help
python3 -m benchmarks.state_bench.v2.summarize_public_systems_formal --help
python3 -m benchmarks.state_bench.v2.audit_morphz_mind_frame_transfer --help

# ME-08：Morphz Runtime 提交 89adf739454da52bce2b35b00fb9e8fa050c5557；
# Morphz 对照组评测基础设施提交 f8ba6bbdfacdf59dad8e226c48e26909a6237a9b；
# Codex 对照基础设施提交 a226bfef1b555e2d83fa4b3ce6d90790bc522705
# 最新修复版 Morphz 与冻结 Codex 派生对照证据提交 63a437e；
# Codex 超时对称复核证据提交 4bcc79f；
python3 -m benchmarks.harbor.run_benchmark --help
python3 -m benchmarks.harbor.run_codex_comparison --help
python3 -m benchmarks.harbor.run_me08_postfix_all89_morphz --help
python3 -m benchmarks.harbor.summarize_me08_full89_pair --help

# ME-08 缓存传输 A/B：Runtime 提交 89adf739454da52bce2b35b00fb9e8fa050c5557；
# 冻结证据提交 011cec8；验证器复评证据提交 b38a3a70359374e260308d2e6466ed61f66b0251；
# 两组策略、二进制哈希与复评结果记录在冻结清单中
```

复现实验时应固定仓库提交版本、模型标识、模型服务配置、随机种子或不支持随机种子的事实、原始JSON、评分器版本和二进制哈希；发布包不应依赖本机临时路径或包含凭据。

参考文献

[1] John McCarthy. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. *Communications of the ACM*, 3(4):184–195, 1960. https://doi.org/10.1145/367177.367199

[2] Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. ReAct: Synergizing Reasoning and Acting in Language Models. *ICLR*, 2023. https://arxiv.org/abs/2210.03629

[3] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, et al. Toolformer: Language Models Can Teach Themselves to Use Tools. *NeurIPS*, 2023. https://arxiv.org/abs/2302.04761

[4] Luyu Gao, Aman Madaan, Shuyan Zhou, et al. PAL: Program-Aided Language Models. *ICML*, 2023. https://arxiv.org/abs/2211.10435

[5] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *TMLR*, 2023. https://arxiv.org/abs/2211.12588

[6] Zhoujun Cheng, Tianbao Xie, Peng Shi, et al. Binding Language Models in Symbolic Languages. *ICLR*, 2023. https://arxiv.org/abs/2210.02875

[7] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Prompting Is Programming: A Query Language for Large Language Models. *PLDI*, 2023. https://arxiv.org/abs/2212.06094

[8] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, et al. SGLang: Efficient Execution of Structured Language Model Programs. *NeurIPS*, 2024. https://arxiv.org/abs/2312.07104

[9] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, et al. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *ICLR*, 2024. https://arxiv.org/abs/2310.03714

[10] Aditya Thimmaiah, Jiyang Zhang, Jayanth Srinivasa, Junyi Jessy Li, and Milos Gligoric. LLMs Lean on Priors, Not Programming Language Semantics. *ICML*, 2026. https://arxiv.org/abs/2510.03415

[11] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *NeurIPS*, 2023. https://arxiv.org/abs/2303.11366

[12] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, et al. Voyager: An Open-Ended Embodied Agent with Large Language Models. *TMLR*, 2024. https://arxiv.org/abs/2305.16291

[13] Yanming Liu, Xinyue Peng, Jiannan Cao, et al. ToolGate: Contract-Grounded and Verified Tool Execution for LLMs. *Findings of ACL*, 2026. https://doi.org/10.18653/v1/2026.findings-acl.470

[14] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. *UIST*, 2023. https://arxiv.org/abs/2304.03442

[15] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. 2023. https://arxiv.org/abs/2310.08560

[16] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive Architectures for Language Agents. *TMLR*, 2024. https://arxiv.org/abs/2309.02427

[17] Kai Mei et al. AIOS: LLM Agent Operating System. *COLM*, 2025. https://arxiv.org/abs/2403.16971

[18] Yi Yu, Liuyi Yao, Yuexiang Xie, et al. Agentic Memory: Learning Unified Long-Term and Short-Term Memory Management for Large Language Model Agents. *ACL*, 2026. https://doi.org/10.18653/v1/2026.acl-long.981

[19] Buqiang Xu, Yijun Chen, Jizhan Fang, et al. StructMem: Structured Memory for Long-Horizon Behavior in LLMs. *ACL*, 2026. https://doi.org/10.18653/v1/2026.acl-short.12

[20] Samuel Holt, Max Ruiz Luyten, and Mihaela van der Schaar. L2MAC: Large Language Model Automatic Computer for Extensive Code Generation. *ICLR*, 2024. https://arxiv.org/abs/2310.02003

[21] Playbooks AI. Language Runtime, PBAsm, Compiler, and Runtime Documentation. Accessed August 4, 2026. https://docs.runplaybooks.ai/

[22] Di Wu, Zixiang Ji, Asmi Kawatkar, Bryan Kwan, Jia-Chen Gu, Nanyun Peng, and Kai-Wei Chang. LongMemEval-V2: Evaluating Long-Term Agent Memory Toward Experienced Colleagues. 2026. https://arxiv.org/abs/2605.12493

[23] Mike A. Merrill et al. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. *ICLR*, 2026. https://arxiv.org/abs/2601.11868

[24] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. 2025. https://arxiv.org/abs/2504.19413

[25] Microsoft. STATE-Bench: Stateful Agent Task Evaluation Benchmark, version 0.8.1. 2026. Accessed August 26, 2026. https://github.com/microsoft/STATE-Bench

[26] The Terminal-Bench Team. Terminal-Bench 2.1: A Revision of Terminal-Bench 2.0. 2026. https://www.tbench.ai/news/terminal-bench-2-1

[27] Edsger W. Dijkstra. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Communications of the ACM*, 18(8):453–457, 1975. https://doi.org/10.1145/360933.360975