熵驱动的记忆重构认知模型

V3

Hongyou Wang
Independent Researcher
zippy618@msn.com

2026

摘要

现有以 Transformer 为核心的大规模语言模型，普遍隐含一个未经检验的前提：只要在给定输入条件下生成概率最高的文本序列，推理过程便可视为完成。在这一范式中，系统被默认要求在任何时刻、任何问题下都进入表达阶段，而“是否已经具备回答条件”并未作为一个显式、可计算的对象加以建模。大量实践表明，这一假设正是语义漂移、自相矛盾、表面合理但不可验证结论等问题的根源之一，而这些问题无法仅通过扩大模型规模或提升生成精度得到根本缓解。

与此形成对照的是，人类在面对复杂、不确定或信息不足的问题时，通常具备一种前语言层面的判断能力：意识到“当前尚未想清楚”，从而选择继续思考、补充结构或暂缓表达。这种能力并不等价于答案正确性的评估，也不依赖显式逻辑符号推演，而是一种对自身认知状态是否已收敛至可表达区域的判断。本文的核心目标，正是将这一能力形式化，使其成为认知系统中的一个显式机制。

本文提出一种以**可表达性熵**为核心判据、以**记忆重构（REWRIT）**为动力的认知推理模型。该模型不以“生成什么答案”为推理目标，而是将推理重新定义为：在给定查询约束下，通过结构加工推动系统内部状态进入一个风险可控、结构稳定、可被可靠表达的区域。语言生成仅被视为这一过程完成后的表达阶段，而非驱动推理本身的核心机制。

在模型中，系统状态由查询（query）与有限候选记忆集合共同构成，答案文本被明确排除在系统状态之外。为判断当前状态是否具备进入表达阶段的资格，本文引入可表达性熵这一结构性风险度量。该熵并非热力学熵或香农信息熵，而是一种面向认知与工程的状态不确定性刻画，综合反映三类风险来源：候选记忆对查询语义的覆盖不足、记忆之间的内部冲突，以及在历史路径中反复失败所体现的结构不稳定性。系统仅在熵值低于给定阈值 ε时，才被允许进入语言生成阶段。

当系统状态尚未进入可表达域时，模型触发 System 2 结构推理过程。System 2 并不生成答案文本，其唯一职责是通过 REWRITE 操作引入新的记忆结构，使整体状态在熵意义上严格下降。REWRITE 不要求逻辑完备或语义完整，其评价标准仅在于是否降低了可表达性熵。为保证推理路径的稳定性与可追溯性，模型施加单调扩展约束：系统禁止删除或回溯既有记忆，只允许通过新增或重构结构来推进认知状态。

在长期运行中，有效的 REWRITE 模式可通过虚拟环境训练被抽象和沉淀为可复用的经验结构，从而逐步前移为快速检索阶段（System 1）即可调用的认知模板。为防止记忆无序膨胀，本文进一步引入记忆熵守恒约束，要求系统在持续学习与写回的过程中，总体记忆资源保持有界，通过抽象与压缩而非简单累积实现能力提升。这一约束保证了模型在长期运行下的容量稳定性、推理效率与结构自洽性。

从整体上看，本文模型在结构上严格区分检索、推理与表达三个阶段，并通过熵判别器实现阶段切换，使“是否该回答”成为一个显式、可调节、可学习的对象。该框架并不依赖特定的语言模型、检索机制或环境形式，因而可作为一种更底层的认知约束层，与现有生成模型、RAG 系统或执行模块协同使用。

本文不主张该模型立即替代现有大规模语言模型，而是提出一种不同的认知出发点：推理的本质不在于生成答案，而在于让系统状态变得可以被稳定表达。在这一视角下，语言生成只是认知过程的最后一步，而非承担认知判断与风险控制的核心角色。该框架为构建更稳健、可控、具备长期演化能力的智能系统提供了一种新的结构路径。

为避免“仅靠自洽叙述而不可检验”的风险，本文进一步给出一套可工程落地的验证协议：系统在每次推理中输出可审计的结构轨迹（S1 检索集合、S2 每步新增结构、熵分量与停止原因），并以“是否允许表达（answer/refuse）”与“证据引用是否合规（memory id / claim key）”作为主要评分对象，而不直接评价自由文本答案质量。该协议允许与任意 Transformer 模型在同一任务集上进行 **决定+引用** 形式的对比，从而将差异集中到“是否过早进入表达”与“冲突是否被调解”等结构性指标上。

引言：为什么“能不能回答”比“怎么回答”更重要

问题动机

传统生成范式把“推理完成”等价为“生成概率最高的文本”。在工程实践中，这会导致系统在结构尚未成熟时被迫进入表达，从而出现语义漂移、自相矛盾、表面合理但不可验证的结论。本文的立场是：推理的本质不在于立即生成答案文本，而在于推动系统内部结构收敛到一个风险可控、可被稳定表达的区域；语言生成属于后置的表达阶段。

一个工程视角的直觉。

如果把“生成文本”当作系统唯一出口，那么系统就会被迫在任何输入下给出某种输出；而真实系统中最难的往往不是“把话说得像样”，而是“此刻是否应该说”。尤其在以下场景里，生成式系统的风险会显著放大：

- **信息不足：**问题缺少关键事实，但系统仍倾向给出完整结论（导致不可验证或误导性的输出）。
- **内部矛盾：**检索到的候选证据彼此冲突，系统在没有调解机制时只能“拼凑一个看起来顺”的叙述。
- **历史反复失败：**某类结构组合在过去多次走不通，但系统每次仍会以类似方式进入表达（表现为震荡、循环论证、反复改口）。

这些问题的共同点是：它们不是“语言组织能力不足”，而是“**表达资格判定缺失**”。

本文要显式建模的核心问题。

本文关注的是一个可操作的工程问题：在不预设形式逻辑体系、也不依赖答案监督的前提下，系统如何判断当前状态是否已经成熟到**允许进入表达阶段**？换句话说，系统能否像人类一样意识到“我还没想清楚”，并把它变成一个可计算、可调、可验证的对象。

贡献（面向工程读者）。

本文的贡献可以理解为把推理流程拆成“检索–结构推理–表达”三段，并给出一套统一的门控与验证机制：

- 用状态 (q, M) 明确排除答案文本，避免推理被“早期答案承诺”牵引；
- 引入可表达性熵 $E(q, M)$ 作为结构风险标量，并用阈值 ε 定义可表达域；
- 当未进入可表达域时，触发 S2 仅通过 REWRITE 新增结构并强制熵严格下降；
- 给出可审计轨迹与 决定+引用 评测协议，使框架可以被复现、打分和与 Transformer baseline 对比（见第 6.10 节）。

System 1 / System 2 的工程化重定义

本文沿用 System 1 / System 2 的命名，但剥离其心理学隐喻，给出可执行定义：

- System 1（S1）：快速、无迭代、无状态的初始化检索，仅负责召回候选结构；
- System 2（S2）：以“结构成熟度”为目标的显式结构加工过程，不输出答案文本，仅通过 REWRITE 新增结构使熵严格下降。

系统何时从 S1 进入 S2、何时从 S2 进入表达，不由主观“问题复杂度”决定，而由熵判别器统一决定（第 4 节）。

为什么需要一个可计算的切换判据。

在很多“慢思考 prompting / 多步推理”的实践里，S2 仅仅意味着“写得更长、步骤更多”。本文的 S2 不追求更长的文本链条，而追求一个可度量的状态收敛目标：让 (q, M) 进入可表达域。切换判据由熵判别器提供，避免“凭感觉判断是否该继续想”的不可检验性。

基本对象：为什么答案不应作为系统状态的一部分

Query（问题 / 任务）

记 query 空间为 $\mathcal{Q}$，任意一次推理由一个 query 触发：

$$q \in \mathcal{Q}.$$

在本文中，q 是对系统状态的约束条件，而非必须立即映射为答案文本的输入。q 可是自然语言、结构化任务、多模态信号或复合指令；系统围绕 q 构造一个可表达的内部结构状态。

例子：把 query 当作约束，而不是“立即回答的提示词”。

同一句自然语言问题在不同系统里可以代表不同的约束边界：例如“这个结论可靠吗？”可能意味着需要引用可审计证据；“这个方案可执行吗？”可能意味着需要满足若干约束与资源限制。本文将 q 视为**表达门槛的定义者**：它决定什么样的结构状态才算“足以表达”，而不是强迫系统立刻输出某个答案句子。

qv 记忆（记忆原子）

记系统的长期记忆全集为 $\mathcal{M}$。每一条记忆原子为：

$$m_i = (q_i, v_i, z_i, c_i, s_i, \eta_i) \in \mathcal{M}.$$

其中：

- q_i：触发语境/适用上下文（可为分桶签名或结构化条件）；
- v_i：内部结构的自然语言投影载体（便于审计/调试，但不等价于答案文本）；

- $z_i \in \mathbb{R}^d$：向量表示（embedding），仅用于相似度与检索；
- $c_i > 0$：资源/容量成本（不表示正确性或置信度）；
- $s_i \in [-1,1]$：方向权重（支持/抑制），不等价于真值；
- η_i：元数据（冲突键、极性、调解记录、rewrite 来源、bookkeeping 等）。

工程实现通常还会为每个原子附加稳定标识 id_i，用于审计日志、集合签名与引用约束；该句柄不改变本文的形式化状态定义。

一个最小但直观的例子。

为了让冲突与调解在工程上可实现，常见做法是让 v_i 具有可解析的“结构标签”。例如：

```
FACT:city=beijing
NOT:city=beijing
CONSTRAINT: reconcile (m_a, m_b)
```

其中前两条在相同 claim key（city）下极性相反，容易被判为强冲突；第三条不是“新事实”，而是一个**调解结构**，它可以把“这对冲突在当前语境下如何被处理”显式写入 M，从而在不删除任何原子的前提下降低冲突风险（详见 4.5 与 5.4）。

方向权重 s_i 的工程语义。

s_i 不是置信度或真值，它更像“使用倾向”：正值表示鼓励系统使用该结构，负值表示提醒系统谨慎或避免（例如反例、禁忌、陷阱提示）。把这类“负向记忆”纳入同一载体，可以让系统在不引入额外逻辑语言的情况下表达约束性经验。

系统状态

系统在任意时刻的状态定义为：

$$S = (q, M), \qquad q \in \mathcal{Q},\ \ M \subseteq \mathcal{M},\ \ |M| < \infty.$$

需要强调：**系统状态中不包含答案文本。**原因是答案一旦进入状态，系统会过早围绕某一答案形态优化，从而掩盖“当前结构是否已成熟可表达”的核心判定问题。

为什么“排除答案文本”在工程上很关键。

如果答案是状态的一部分，那么推理过程很容易退化为“不断修补一段文本”，而不是“改善结构条件”。这会带来两个常见后果：一是早期文本会对后续推理形成路径依赖（越写越难改）；二是系统会把“说得像答案”误认为“具备回答条件”。本文用 (q, M) 把“结构成熟度”从“语言表述”中剥离，使得门控与推理能够被外部验证。

记号与参数总表

表示函数与向量空间

向量维度 $d \in \mathbb{N}$。定义 query 与记忆的编码函数：

$$\phi_Q : \mathcal{Q} \to \mathbb{R}^d, \qquad z_q = \phi_Q(q),$$
$$\phi_M : \mathcal{M} \to \mathbb{R}^d, \qquad z_i = \phi_M(m_i).$$

本文不限定其训练方式或模型结构；唯一要求是可比性：输出向量位于同一空间用于相似度计算。

工程实现建议：表征层只提供几何相似度。

第 0 层（表征层）在概念上与“LLM 的分词/编码”阶段类似：把自然语言投影到一个向量空间中。但本文强调它**只负责几何相似度**，不承担“是否可表达”“是否可信”等判断责任。工程上可以用 Transformer embedding、也可以用轻量的 hashing bag-of-words；只要相似度可比较即可。关键是：推理逻辑不在表征层，而在后续的熵判别与结构推理（S1/S2）。

相似度与冲突判定

余弦相似度：

$$\cos(u, v) = \frac{u^\top v}{\| u \|_2 \| v \|_2} \in [-1, 1].$$

冲突指示函数：

$$\chi(m_i, m_j) \in \{0, 1\}.$$

当 $\chi = 1$ 表示两条记忆被认为存在强冲突；χ 不要求形式逻辑完备，只回答工程问题：并置是否显著增加结构不确定性。

一个最小可实现的冲突规则。

为了让 χ 可工程化，常见做法是从 η_i 或可解析的 v_i 中提取结构键（claim key）与极性（polarity）：

- 若两条记忆具有相同 claim key，但 polarity 相反，则视为强冲突（$\chi = 1$）；
- 若存在显式约束结构声明“该冲突对已被调解”，则对该对记忆置 $\chi = 0$（不需要删除任何原子）。

这为后续“新增 constraint 记忆以降低冲突熵”提供了工程闭环（见 5.4 与 6.10）。

熵分解权重与阈值

$$\alpha, \beta, \gamma \geq 0, \qquad \alpha + \beta + \gamma = 1, \qquad \varepsilon > 0.$$

其中 ε 为可表达阈值：越小越保守（更稳健但更易拒答），越大越激进（更快但风险更高）。

调参直觉：把它当作风险偏好旋钮。

(α, β, γ) 决定系统更在意哪类风险：

- α 大：更在意“相关性/覆盖”，更像信息检索系统；
- β 大：更在意“内部一致性”，更像一致性审查器；
- γ 大：更在意“历史可达性”，更像保守的经验系统（但会更路径依赖）。

ε 则决定“允许表达”的保守程度：ε 越小越难过门，但更稳健；ε 越大越容易过门，但更像传统生成系统。

迭代、容量与环境参数

S2 最大推理步数：$T_{\max}$。长期记忆容量约束（记忆熵守恒的预算）：

$$S(\mathcal{M}) = \sum_{m_i \in \mathcal{M}} c_i \leq C.$$

环境经验筛选阈值：ε_{env}（用于决定虚拟环境轨迹是否允许写回）。

工程上为什么一定要有 $T_{\max}$。

即便有熵下降约束，实际实现中仍需要最大步数上限来避免极端情况下的长时间搜索；更重要的是，上限让“推理成本”成为可控资源，便于在不同场景下调整延迟/成本与稳健性的权衡。

熵判别器（Entropy Discriminator）：结构成熟度的形式化判定

熵判别器的角色与函数定义

熵判别器回答：在当前 q 与候选结构 M 下，系统是否具备进入表达阶段的资格？

$$E\colon \mathcal{Q} \times \mathcal{P}_{\mathrm{fin}}(\mathcal{M}) \to \mathbb{R}_{\geq 0}.$$

重要约束：E 的输入不包含答案文本或语言生成结果；它评估的是结构状态，而非表述质量。

为什么必须把“是否可表达”做成显式函数。

如果“是否该回答”仅隐含在生成分布里，那么系统就缺少一个可检查、可调参、可学习的门控接口：你无法在工程上回答“为什么这次该拒答？”或“把系统调得更保守应该改哪个参数？”。把门控显式化为 $E(q, M)$ 的好处是：它把表达资格变成一个可度量对象，允许你用实验去检验它是否真的在降低风险，而不是只停留在叙述层面的合理性。

可表达性熵的认知含义

$E(q, M)$ 不是热力学熵或 Shannon 熵，而是面向认知与工程的结构风险刻画：在 q 的约束下使用 M 进入表达阶段时，系统将面临的结构性风险。本文聚焦三类风险：覆盖不足、内部冲突、历史路径不稳。

它衡量的是“表达风险”，不是“信息量”。

直观地说，E 越大，表示系统越可能在表达阶段出现“答非所问、证据拼凑、前后矛盾、反复震荡”等失败模式；E 越小，表示系统越接近一个可被稳定投影为语言

的结构状态。这里的“稳定”不等价于“正确”，但它意味着：在给定约束下，系统至少具备更一致、更可审计的表达前提。

熵函数的分解形式与权重

$$E(q,M) = \alpha\, E_{\mathrm{cov}}(q,M) + \beta\, E_{\mathrm{conf}}(M) + \gamma\, E_{\mathrm{stab}}(q,M).$$

工程上建议对各分量做归一化或单调裁剪，使其位于 $[0,1]$ 区间以便线性加权。

如何理解 E 的三个分量。

你可以把 E 看作一个“风险面板”：

- E_{cov} 高：当前候选集合与 query 的几何相关性不足，容易“无从下嘴”；
- E_{conf} 高：候选集合内部矛盾大，容易“拼凑式自洽或自相矛盾”；
- E_{stab} 高：即便看起来相关且一致，但历史上经常走不通，意味着隐含风险或缺口。

工程上不必执着于唯一形式，只要满足：数值更大代表风险更高，并且对 S2 提供可优化的方向即可。

覆盖熵：相关性不足的度量

设 $z_q = \phi_Q(q)$，对任意 $m \in M$ 设其向量表示为 z_m。定义最大相关性：

$$\mathrm{sim}_{\max}(q,M) = \begin{cases} \max\limits_{m\in M} \cos(z_q, z_m), & M \neq \emptyset, \\ 0, & M = \emptyset, \end{cases}$$

覆盖熵：

$$E_{\mathrm{cov}}(q,M) = 1 - \mathrm{sim}_{\max}(q,M).$$

若希望保持非负、单调的工程得分，可使用裁剪余弦 $\cos^+(u,v) = \max\{0, \cos(u,v)\}$。

为什么用 max 而不是平均。

覆盖熵的基本直觉是：只要候选集合里**存在**一条足够相关的结构，就可能为表达提供支点；因此使用最大相似度更符合“是否有抓手”的门控语义。平均相似度更适合评估“整体相关”，但会把少数关键证据淹没在大量无关记忆里。工程上也可使用 top-k 平均或 softmax 近似，原则是保持单调性与可解释性。

边界例子。

若 $M = \varnothing$，则 $\mathrm{sim}_{\max} = 0$，因此 $E_{\mathrm{cov}} = 1$，表示完全无覆盖；若存在某条记忆与 query 高度相似（$\cos \approx 1$），则 $E_{\mathrm{cov}} \approx 0$。

冲突熵：结构内在矛盾的度量

设 $|M| = n$，定义：

$$E_{\mathrm{conf}}(M) = \begin{cases} \dfrac{2}{n(n-1)} \displaystyle\sum_{i<j} \chi\,(m_i, m_j), & n \geq 2, \\ 0, & n < 2. \end{cases}$$

其直觉是：强冲突对越多，结构风险越高。χ 可由规则/学习分类器/结构一致性校验实现。

一个可计算的小例子。

设 $M = \{m_1, m_2, m_3\}$，其中 (m_1, m_2) 被判为冲突、其余不冲突，则冲突对数量为 1，$n = 3$：

$$E_{\mathrm{conf}}(M) = \frac{2}{3 \cdot 2} \cdot 1 = \frac{1}{3}.$$

若两对冲突（例如 (m_1, m_2) 与 (m_2, m_3)）则为 $2/3$。这个分量的意义是把“内部矛盾程度”变成一个可优化的标量。

冲突调解为什么能降熵。

在单调扩展约束下你不能删除冲突记忆，因此需要一种“把冲突处理方式写进结构”的机制：例如新增 constraint 记忆声明某对冲突在当前语境下如何被调解（或被禁止共同使用），使得 χ 对该对冲突不再计入，从而 E_{conf} 下降（见 5.4 与第 6.10 节）。

稳定性熵：历史路径风险的度量

系统维护历史映射：

$$H(\text{cluster}(q), \text{sig}(M)) \mapsto p_{\text{succ}} \in [0,1],$$

其中 $\text{cluster}(q)$ 为 query 分桶，$\text{sig}(M)$ 为集合签名，p_{succ} 为有限步内达到 $E(q,M) \le \varepsilon$ 的经验成功率。定义：

$$E_{\text{stab}}(q,M) = 1 - p_{\text{succ}}.$$

无历史记录则用先验 $p_0 \in (0,1)$。一次推理完成后用指数滑动平均更新：

$$p_{\text{succ}} \leftarrow (1-\eta)\, p_{\text{succ}} + \eta \cdot \mathbb{I}[\text{success}].$$

为什么需要稳定性分量。

仅靠“当前覆盖”和“当前冲突”可能会让系统反复走进同一种死胡同：在当下看起来相关且一致，但实际推理总是卡在缺口（例如总缺某个关键证据、或总在某一步无法找到可行 rewrite）。E_{stab} 通过历史成功率把这种“结构模式的风险”显式反馈给门控与 S2，让系统逐渐学会避开高风险结构，或更早触发不同的重构方向。

更新规则的直观解释。

如果某个 $(\mathrm{cluster}(q), \mathrm{sig}(M))$ 模式经常成功，则 p_{succ} 会被推高、E_{stab} 变小；反之则变大。η 控制“新经验”对历史的影响强度：η 大则更敏感，η 小则更稳健但收敛慢。

可表达域与停止条件

定义可表达域：

$$R_\varepsilon = \{(q, M) \mid E(q, M) \le \varepsilon\}.$$

若 $(q, M) \in R_\varepsilon$，系统停止 S2 并进入表达阶段。

拒答的语义。

当系统在 $T_{\max}$ 步内找不到任何满足“严格降熵”的 REWRITE，或者外部验证器（第 6.10 节）认为缺乏最低证据条件时，系统选择拒答。这里的拒答不是“模型不会写答案”，而是一种风险控制：系统明确告诉你“在当前结构约束下无法把状态推入可表达域”，从而避免输出不可审计的结论。

熵判别器与“熵记忆”的关系

熵判别器可维护辅助记忆（例如式 [eq:stability-map-cn] 的成功率表）以学习“哪些结构值得继续推理、哪些结构应尽早放弃/修正”。这种学习不构成答案监督；其对象是结构路径的风险与稳定性。

关于 $\mathrm{cluster}(q)$ 与 $\mathrm{sig}(M)$ 的工程取舍。

$\mathrm{cluster}(q)$ 的作用是把 query 粗粒度分桶，避免对每个句子都单独记忆历史；$\mathrm{sig}(M)$ 的作用是把集合模式变成顺序无关的键（例如按 id 排序后哈希）。签名是否包含 constraint/abstraction 原子属于工程选择：包含它能区分“已调解/未调

解”的结构差异，不包含它能提高泛化但会丢失细节。本文强调的是：无论选择哪种签名，只要定义清晰、可复现即可。

S2 推理与记忆重构（REWRITE）：以熵下降为目标的结构加工过程

S2 的定位：它不是“慢速答案生成”

S2 的唯一职责是：当结构未进入可表达域时，通过新增结构加工降低 $E(q, M)$，而不是“更复杂的文本生成”。

与常见“多步推理文本链”的区别。

很多系统把 S2 理解为“写更多思维链/步骤”，但这仍然是在语言空间里做优化。本文的 S2 是在**记忆结构空间**里做状态演化：它不输出答案句子，不追求把逻辑写得漂亮，而追求让 $E(q, M)$ 严格下降，直至状态进入可表达域。这样做的好处是：S2 的每一步都是可审计的结构变化，并且可以被外部验证器约束，避免“写得很像推理但实际上没有证据”的自嗨。

S2 的输入、输出与允许动作

输入

在步 t，S2 接收 (q, M_t) 以及当前熵 $e_t = E(q, M_t)$，并满足前置条件 $e_t > \varepsilon$。

输出

S2 输出新增结构集合：

$$\Delta M_t = \{m_1, \dots, m_k\}, \qquad k \geq 0.$$

S2 允许输出空集（表示找不到可行降熵方向），但不输出答案文本。

输出是什么（以及不是什么）。

ΔM_t 的每个元素都是一个新的记忆原子（同一类型对象），例如桥接结构、冲突调解结构、抽象结构、约束结构等。它们的共同点是：它们不是“最终答案”，而是为了让整体结构更可表达而添加的支撑件。系统把这些新结构合并回 M_t，再重新计算熵，直到允许表达或失败终止。

允许动作集合（Action Space）

S2 允许的唯一操作是 REWRITE：只新增、不删除、不回滚、不直接修改既有记忆内容。

为什么禁止删除/回滚。

禁止删除不是哲学姿态，而是工程约束：它让推理路径具备可追溯性（你可以看到系统到底加了什么结构才过门），也避免了“通过擦除不利证据来获得一致性”的不良行为。若确实需要“弱化某些结构”，可以通过方向权重、抽象覆盖标记或约束结构来实现，而不是直接删除。

单调扩展原理

$$M_{t+1} = M_t \cup \Delta M_t.$$

单调扩展带来的两个好处。

- **可审计：**每一步只会新增结构，因此可以完整复现“系统是靠哪些新增结构从不可表达变为可表达”的路径。
- **可调试：**如果系统过门但结果不可信，能够定位是哪一类 REWRITE 让它过门，从而调整对应策略或外部验证器规则。

REWRITE 的形式化定义

熵下降约束（硬约束）

$$E(q, M_t \cup \Delta M_t) < E(q, M_t).$$

REWRITE 的语义角色

REWRITE 不要求逻辑完备或语义完备；其工程评价标准仅在于是否严格降熵。它可以是桥接、调解约束、抽象压缩等结构单元。

三类典型 REWRITE（工程上最常见）。

- **桥接（bridge）**：把 query 的核心约束用更“可检索”的形式重述，帮助覆盖分量下降；但 bridge 不应被当作证据（需被外部验证器约束）。
- **调解（constraint）**：针对已检测到的冲突对，新增一个“冲突如何被处理”的约束原子，使冲突对不再被计入强冲突，从而降低 E_{conf}。
- **抽象（abstraction）**：当候选集合里出现大量相似结构时，引入抽象原子减少冗余与干扰，使结构更稳定、容量更可控。

这三类并不是唯一动作空间，而是为了演示“仅靠新增结构也能驱动熵严格下降”的最小集合。

REWRITE 的两类来源（逻辑产生的位置）

逻辑模板驱动的 REWRITE（经验型）

若存在可复用模板 ℓ，则可直接产生新结构 $m_{\text{new}} = \ell(q, M_t)$，代表历史上验证有效的降熵方向。

为什么需要模板模式。

如果每次都靠结构搜索，推理成本会很高；更重要的是，很多有效的降熵动作在统计上会反复出现（例如某类冲突的调解方式、某类缺口的桥接方式）。模板模式允许把“过去验证过的降熵方向”沉淀为可复用经验，从而把成本前移到更快阶段。

探索性 REWRITE（结构搜索型）

若无模板匹配，则在受限候选集 $C(q, M_t)$ 中搜索：

$$\Delta M_t \approx \arg\min_{\Delta M \in C(q,M_t)} E(q, M_t \cup \Delta M),$$

不要求全局最优；满足局部严格下降即可。

搜索空间必须受工程约束。

结构搜索不可能无限展开：候选数量、深度、每步允许新增的原子数量都需要上限。本文强调的不是“找到全局最优结构”，而是“在严格下降约束下找到可行方向并在有限步内终止”。

S2 的失败情形与终止

S2 终止条件包括：达到可表达域、达到 $T_{\max}$、或无可行 ΔM_t（所有候选均不满足式 [eq:entropy-descent-cn]）。

无可行 rewrite 的典型原因。

- **缺口不可补：**任务所需的关键 claim key 在长期记忆中根本不存在（或当前 k 的检索永远拿不到），即便加 bridge 也无法满足外部证据条件；
- **冲突无法调解：**冲突判定过于严格或缺少可用调解结构，导致无法降低 E_{conf}；

- **历史不稳定**：稳定性分量主导且历史显示该结构几乎总失败，系统被迫更保守或更早拒答。

S2 输出与长期记忆的关系

新增结构在短期用于推动当前状态降熵；在长期可被写回到 $\mathcal{M}$ 并形成可复用经验，从而逐步前移到 S1 阶段调用。

短期与长期的分工。

短期里，ΔM_t 只需要在当前任务下有效；长期里，只有在多次验证后才值得写回（见 6.7 与 7.5）。这种“先在短期试错，再在长期沉淀”的机制避免了把一次性的噪声结构直接固化进长期记忆。

为什么 S2 必然收敛或终止

在单调扩展、熵严格下降、下界 $E \geq 0$ 与步数上限 $T_{\max}$ 共同约束下，S2 不可能无限循环：要么进入可表达域，要么终止并拒答。

小结：S2 的本质

S2 是“把系统推向可表达域”的结构加工机制，其成功标准是结构风险下降，而非文本相似度提升。

主流程：从输入到表达的完整推理路径

推理问题的形式化描述

一次推理任务的目标不是立即输出答案，而是构造一个有限集合 M^* 使得 $(q, M^*) \in R_\varepsilon$。

成功与失败的工程定义。

成功并不等价于“答案正确”，而等价于“系统状态进入可表达域并满足门控条件”；失败并不等价于“系统能力差”，而等价于“在约束下无法把状态推入可表达域”。这种定义把系统的第一目标从“说出某句话”转为“是否具备表达资格”，更适合做风险控制与可验证推理。

阶段一：S1 初始化检索（快速联想）

定义检索算子：

$$S_1 : \mathcal{Q} \to \mathcal{P}_{\text{fin}}(\mathcal{M}), \qquad M_0 = S_1(q).$$

S1 仅负责召回，不优化熵、不解决冲突、不判断表达资格。

S1 的一个工程实现。

最常见实现是向量相似度 top-k 检索，并可用方向权重 s_i 影响排序（例如只鼓励使用 $s_i > 0$ 的原子）。这里的关键约束是：S1 不做“结构修复”，它只给出一个起点 M_0。

阶段二：熵判别与阶段切换判定

计算 $e_t = E(q, M_t)$。若 $e_t \le \varepsilon$，进入表达阶段；若 $e_t > \varepsilon$，进入 S2。

示例 1：S1 即可通过（不触发 S2）。

若 M_0 中存在至少一个与 q 高度相关的事实结构，且内部冲突很低、历史上也较稳定，则 $E(q, M_0)$ 可能直接落入 R_ε。此时系统不会进入 S2，而是直接把 M_0 投影为输出。工程上这对应“常识性、信息完备、证据一致”的问题。

阶段三：S2 结构推理循环（REWRITE 驱动）

```
t <- 0
M0 <- S1(q)

while (E(q, Mt) > eps) and (t < Tmax):
   DeltaMt <- R(q, Mt)          # REWRITE proposal
   if DeltaMt is empty:
      break                     # failure termination
   Mt+1 <- Mt union DeltaMt
   t <- t + 1
```

示例 2：触发 S2 并通过调解降熵。

若 M_0 内部存在强冲突（例如同一 claim key 的正反事实同时出现），则 E_{conf} 会显著抬高总熵，系统触发 S2。S2 可以新增一个 constraint 原子，显式声明某对冲突在当前语境下如何被调解（或被禁止共同使用），使 χ 对该对冲突不再计入，从而 E_{conf} 下降。此时系统依然没有删除任何原子，只是新增了“冲突处理方式”的结构，使状态更可表达。

阶段四：语言生成（表达阶段）

当 $(q, M^*) \in R_\varepsilon$ 时，调用生成器：

$$G: \mathcal{Q} \times \mathcal{P}_{\text{fin}}(\mathcal{M}) \to \mathcal{Y}, \qquad y = G(q, M^*).$$

生成器不参与 E 的计算、不触发 REWRITE，仅负责将结构投影成文本。

为什么表达层要“足够弱”。

如果生成器参与门控或反过来影响 S2，那么系统会重新退化为“为了生成而推理”。本文刻意把表达层做成可替换的投影模块：你可以换成 LLM、模板、甚至移除它，认知核心的判别与结构推理仍然成立。

语言生成器的约束与边界

为保持结构-表达解耦，生成器必须满足：不影响 S1、S2 与熵判别；其失败/幻觉不应污染推理路径。

记忆写回与经验沉淀（可选）

一次推理完成后，可将部分新增结构写回长期记忆库：

$$\mathcal{M} \leftarrow \mathcal{M} \cup \widehat{M}, \qquad \widehat{M} \subseteq (M^* \backslash M_0).$$

写回条件可包括：多次出现、平均显著降熵、通过虚拟环境或真实任务验证。

写回必须可审计。

写回是长期演化的关键，也是风险来源：不加筛选的写回会把一次性的噪声结构固化为长期记忆，导致系统“越用越乱”。因此工程上建议：为每个写回原子记录来源（seed/env/rewrite）、触发条件、在多少次任务中出现、平均降熵贡献、以及是否满足外部验证器条件。这样写回从“累积”变成“可控沉淀”。

主流程的整体性质

- 可终止性：熵严格下降 + 下界存在 + 最大步数约束；
- 可控性：$\varepsilon, T_{\max}, \alpha, \beta, \gamma$ 等显式控制风险偏好；

- 模块解耦：检索、推理、表达三阶段互不污染；
- 可演化性：有效 REWRITE 可沉淀为经验并前移到 S1。
-

小结

主流程把“能不能回答”变成显式判定对象：系统先判别结构是否可表达，再用结构推理降低风险，最后才进入表达。

工程化验证：可审计轨迹与可打分评测协议（新增）

上述定义是模型无关的，但若缺少验证接口，容易产生“看似降熵但不可外部检验”的退化机制（例如 bridge 复述 query 让覆盖熵下降）。本节给出一个最小、客观、可复现的工程协议，使框架可被打分与比较。

轨迹输出（Trace Schema）

对每个 q 输出机器可读 trace，至少包含：参数（$k, \varepsilon, T_{\max}, \alpha, \beta, \gamma$）、S1 的 M_0（含检索排序/相似度）、S2 每步的熵分量与新增结构、以及终止状态与原因。trace 使得单调扩展与熵严格下降约束可被自动检查。

为什么必须输出 trace。

如果没有 trace，“No-DROP”和“严格下降”就只能停留在口头约束：你无法验证系统是否真的遵守，也无法在出现问题时定位是哪一步的哪类重构导致过门。trace 把推理变成工程对象：可复现、可对比、可回放。

一个最小 JSON 结构示例（仅示意）。

```
{
  "q": "...",
  "config": {"k":5,"epsilon":0.35,"tmax":6,"alpha":0.5,"beta":0.3,"gamma":0.2},
  "s1": {"retrieved_ids":["m1","m2","m3"], "scores":[0.42,0.31,0.12]},
  "steps": [
    {"t":0,"E":0.61,"E_cov":0.40,"E_conf":0.50,"E_stab":0.50,"rewrite":null},
    {"t":1,"E":0.42,"E_cov":0.38,"E_conf":0.20,"E_stab":0.50,
     "rewrite":"constraint","added_ids":["rw_constraint_1"]}
  ],
  "decision": {"success":true,"reason":"E<=epsilon"}
}
```

外部验证器 $V(q, M)$（阻止 bridge-only 过门）

引入外部、确定性验证器：

$$V\colon\ \mathcal{Q} \times \mathcal{P}_{\mathrm{fin}}(\mathcal{M}) \to \{0,1\}.$$

工程评测中，系统仅在 $E(q, M) \le \varepsilon$ 且 $V(q, M) = 1$ 时允许进入表达。一个保守可实现设计：

- 证据条件：M 中必须存在至少一个**非合成**原子匹配任务所需 claim key（例如可解析的 FACT:key=value），且与 q 相似度高于阈值；
- 冲突条件：若 M 中存在未被显式调解的强冲突对，则 $V(q, M) = 0$。

其中“非合成”可由元数据 $\eta_i.\mathrm{source} \in \{\text{seed,env,rewrite}\}$ 标记，并在证据条件中排除 rewrite。

验证器的核心目的：把“证据”与“自我补丁”分开。

bridge 能降低覆盖风险，但它本质上是“结构对齐/索引辅助”，不是独立证据。如果没有验证器，系统可能通过不断新增 bridge 让 E_{cov} 很低，从而过门并输出

看似合理的结论。$V(q, M)$ 的证据条件强制系统至少引用一个非合成事实/约束结构，使“过门”与“有证据”绑定，从而把模型从自嗨式降熵推向可审计降熵。

任务与指标（Tasks & Metrics）

定义任务实例：

$$\tau = (q, \varepsilon, T_{\max}, \mathcal{M}, \text{expected}),$$

expected 只规定可客观打分的结构期望（express/refuse、是否触发 S2、必须出现的 claim key / memory id、冲突场景下必须有调解结构等）。建议报告：pass rate、unsafe answer rate（初始门控失败却表达）、over-refusal rate（存在可行降熵路径却拒答）、以及 S2 行为统计。

为什么不直接评价自由文本答案质量。

如果评测目标是“文本好不好”，就会被写作风格与措辞主导；而本文要验证的是结构风险控制（是否过早表达、是否在冲突下保持安全、是否遵守证据约束）。因此评测应尽量收敛到客观字段：决策与引用，这样才能公平比较不同系统。

任务类型建议（用于压力测试）。

为了让门控与 S2 的作用可见，任务集建议包含至少三类：

- **可直接表达：**S1 就能拿到关键事实且无冲突；
- **缺关键事实：**无论怎么 rewrite 都无法满足证据条件，应拒答；
- **可调解冲突：**S1 能检索到冲突对，S2 新增约束调解后才允许表达。

Baseline 对比：Transformer 对比 决定+引用

为避免纠缠于“自由文本写得好不好”，baseline 协议采用 决定+引用：对每个任务只输出二值决策（answer/refuse）与引用集合 used_ids。若 answer，则 used_ids 必须非空且只能来自该任务的 M_0，从而把对比集中到结构性风险控制能力上。

决定+引用 的输出示例。

{"task_id":"t042","refuse":false,"used_ids":["m12","m15"],"reason":""}

当 baseline 选择回答但没有引用、或引用了不在 M_0 中的 id，评测应判为不合规。这迫使 baseline 在“回答冲动”与“证据约束”之间做权衡，从而更公平地比较表达门控能力。

参考跑数（Reference run）

在一个包含 100 个 pitfalls 任务的合成套件中（缺事实诱导 / 强冲突 / 引用约束），认知核心参考跑数为 $100/100$；以 DeepSeek-V3.2 作为外部 Transformer baseline 的参考跑数为 $88/100$，其安全式通过率为 $78/100$。上述数字用于说明协议可产出外部可检验度量，不代表泛化结论。

虚拟环境（Virtual Environment）：在低成本条件下学习降熵模式

为什么需要虚拟环境

主流程高度依赖 S2 的 REWRITE 是否能持续降熵；但真实任务分布稀疏且昂贵，且降熵信号通常是延迟的。因此需要一个可控、可重复、低成本的试验空间批量产生“结构–熵–演化轨迹”，以学习哪些结构变化更可能有效。

三个现实问题。

如果只靠线上真实任务驱动学习，通常会遇到：

- **真实问题昂贵且稀疏：**高价值 query 出现频率低、失败代价高，不适合作为探索场；
- **降熵是结构信号：**如果训练直接绑定答案监督，很容易把“表达成功”误学为“文本相似度最大”；
- **学习信号延迟：**某次 REWRITE 是否有效往往要在多步熵演化后才能判断。

虚拟环境的目的就是把“结构探索”转移到低成本空间里，让系统能以更高频率获取降熵经验。

虚拟环境的角色定位

虚拟环境不是世界仿真器；它不要求生成真实答案，只要求生成足够多的状态–动作–结构变化序列，使系统学习“哪些结构变化在统计上更容易降熵”。

与“模拟现实世界”无关。

这里的环境可以非常抽象：它甚至可以只生成冲突对、缺口、约束图等结构对象。关键不是“像不像现实”，而是能否产生足够多的结构–熵轨迹，让系统学到可复用的降熵方向。

虚拟环境的形式化定义

虚拟环境定义为四元组：

$$\mathcal{E}_{\mathrm{env}} = (X, A, f, g),$$

其中 X 为环境状态空间，A 为动作空间，$f: X \times A \to X$ 为转移函数，$g: X \to O$ 为观测映射；不要求 f 可逆或确定。

一个直观解释。

你可以把 X 理解为“结构状态”（例如约束集合、关系图、候选记忆子图），把 A 理解为“结构操作”（合并、条件化、桥接、抽象、调解）。环境只需要提供一种方式让这些操作发生，并能观测到熵是否下降。

从环境轨迹到推理经验

环境生成轨迹 $\tau = (x_0, a_0, \ldots, x_T)$。轨迹不直接进入推理系统，而经抽象算子生成经验记忆：

$$G_{\mathrm{env}}: \tau \to m_{\mathrm{env}}, \qquad m_{\mathrm{env}} = \langle q_{\mathrm{env}}, v_{\mathrm{env}}, z_{\mathrm{env}}, c_{\mathrm{env}}, s_{\mathrm{env}}, \eta_{\mathrm{env}} \rangle.$$

其意义是“某类结构条件下，某种重构方向往往有效”，而非“答案是什么”。

经验写回的含义。

写回到长期记忆的不是“某题答案”，而是“在某类结构条件下，采取某种结构动作往往能降熵”的经验单元。这样 S2 的有效模式可以逐步沉淀并前移，让未来更多任务在 S1 阶段就能检索到更接近可表达域的结构起点。

环境熵与经验筛选（防止噪声污染）

并非所有轨迹都值得写回。引入环境熵 $E_{\mathrm{env}}(\tau)$，以 $E_{\mathrm{env}}(\tau) \le \varepsilon_{\mathrm{env}}$ 作为写回条件，从而过滤噪声经验。

筛选的工程意义。

如果不筛选，虚拟环境会把大量偶然有效、不可复用的噪声模式写回，导致长期记忆膨胀并污染检索。环境熵的目标是：只保留那些“结构演化稳定、降熵趋势持续、可被压缩为低复杂度经验”的轨迹。

训练目标：学习熵下降能力而非答案

训练目标是最大化结构性降熵能力：

$$\max \mathbb{E}_{q,M}[E(q,M) - E(q,M \cup \{m_{\mathrm{env}}\})].$$

与传统监督训练的区别。

传统训练常把目标绑定到“输出文本/标签”，而这里的目标绑定到“结构风险下降”。这使得框架可以与任意生成模型协同：即使没有强答案监督，系统仍然可以通过结构轨迹积累经验，逐渐学会更有效的重构方向。

训练如何反哺主流程

被验证有效的经验可迁移到主流程：丰富 S1 可检索结构、增强 S2 模板库，从而把成本前移到更快阶段。

两个反馈路径。

- **强化 S1**：有效经验进入长期记忆后，S1 更可能直接检索到“近可表达”结构，减少 S2 次数；
- **强化 S2**：经验作为模板让 S2 先尝试高成功率的重构方向，降低搜索成本与失败率。

训练与推理的严格解耦

虚拟环境训练不要求绑定答案监督；其产物是结构性经验记忆与降熵策略，而非语言生成器权重。

为什么必须解耦。

如果训练目标与答案绑定，系统会自然倾向于优化“说得像答案”，再次把推理拉回到表达空间。严格解耦能让系统把学习资源用在“结构成熟度”上，并通过门控决定何时才进入表达。

小结：为什么系统可以“越用越会想”

若经验写回与筛选有效，S2 的降熵模式会逐步沉淀，未来更多 query 可在 S1 阶段直接召回近可表达结构，从而减少 S2 迭代。

记忆熵守恒：长期认知系统的稳定性约束

问题动机：为什么“记得越多”反而是风险

持续写回若不加约束，会导致冗余与干扰累积，使检索与推理成本失控。风险不在条目数本身，而在资源占用与维护复杂度的无界增长。

为什么这是一个现实工程问题。

一旦系统允许写回，长期记忆很容易出现三个现象：

- **重复**：同义或近义结构大量累积；
- **干扰**：无关/弱相关结构增加检索噪声，使覆盖熵与稳定性恶化；
- **维护成本上升**：冲突对数量随规模上升而迅速增加，导致冲突熵更难控制。

因此，“长期可用性”不仅取决于推理机制，还取决于记忆容量与结构复杂度是否可控。

记忆熵的定义（不是记忆条目数）

用资源成本 c_i 刻画记忆占用，定义：

$$S(\mathcal{M}) = \sum_{m_i \in \mathcal{M}} c_i, \qquad S(\mathcal{M}) \le C.$$

为什么用 c_i 而不是 $|\mathcal{M}|$。

条目数并不能反映真实成本：一条结构化的经验模板可能比十条碎片事实更“重”，也更干扰检索；相反，一些低成本的提示/约束可能几乎不增加检索负担。用 c_i 可以把存储、检索、干扰与维护的工程影响统一到一个预算上。

守恒原则的核心思想

学习必须在有界预算下进行：用更好的结构替代更差的结构，而不是无限追加条目。

守恒不是“停止学习”。

守恒约束并不阻止系统变强，而是迫使学习变得更高效：系统需要通过抽象、压缩、重组把经验变成更通用的结构，而不是简单堆叠记忆碎片。

为什么禁止删除，而允许“方向重写”

禁止删除保证路径可追溯性与稳定性；但允许通过方向权重或元数据标记“暂不使用/被抽象覆盖”，从而在不删除的前提下控制有效复杂度。

工程上的折中。

完全禁止删除的代价是“存储会增长”；因此需要配套的抽象/压缩机制来消化增长。方向重写（例如把被抽象覆盖的原子标记为低使用倾向）提供了一种不破坏可追溯性的“软淘汰”方式。

抽象压缩作为守恒机制

当 $S(\mathcal{M}) > C$ 时触发抽象算子 $\mathcal{A}$，将若干记忆压缩为抽象记忆 m_{abs}，并要求 $c_{\text{abs}} \leq \sum_j c_j$，且尽量保持其对降熵能力的贡献。

一个最小实现思路。

工程上可以把“相似的一组记忆”聚合成一个抽象原子（例如对其 embedding 做平均、对其 claim key 做上位概念归纳），并在元数据里标记哪些原子被其覆盖（subsumed_by）。被覆盖原子不删除，但其 s_i 可被调小（趋近 0），使其在检索与推理中的有效影响降低，从而实现“无删除但可控”的容量管理。

记忆“方向”的概念（方向性记忆）

方向权重 $s_i \in [-1,1]$ 用于表达“支持/避免/陷阱提示”等方向性信息，独立于容量成本 c_i。

负向记忆的用途。

很多系统只存“正向事实”，但工程实践里“不要做什么”同样重要：例如反例、禁忌条件、已知失败模式。方向性记忆允许系统把这些经验纳入结构推理中，同时不必把它们硬编码为逻辑规则。

守恒与学习并不矛盾

守恒迫使系统“记得更好而不是记得更多”，推动抽象与结构效率提升。

记忆熵守恒与人类认知的类比

类比意义上，人类会把碎片经验压缩为更高层概念，而非简单累加细节；守恒机制在工程上模拟这种长期稳定性。

记忆熵守恒对系统整体的影响

引入守恒后，系统具备容量有界、推理效率可控、结构路径不断裂、学习方向更明确等性质。

与 S1/S2 的关系。

守恒机制对 S1 的影响是降低检索噪声与干扰，对 S2 的影响是减少“无效重构”的搜索空间。更重要的是，它让系统可以长期运行：写回不再意味着必然膨胀，而意味着“结构会越来越抽象、越来越可复用”。

小结

记忆熵守恒提供长期稳定性下界：在持续学习/写回条件下仍保持容量可控与结构自洽。

讨论（Discussion）

关于“熵”作为核心判据的合理性

熵将“是否该说”从“说什么”中剥离，提供连续、可比较、可优化的标量目标，使推理具备工程可调性。需要承认：$E(q, M)$ 的具体形式并非唯一，需随任务与风险偏好调整。

它为什么比“让模型多想几步”更工程化。

很多方法通过增加推理步数来降低错误，但“步数”本身不是风险度量：你无法判断“再想一步”是否真的更安全。熵判别器提供的是一个可比较的标量目标：你可以观察 E 是否下降、哪一项在下降、下降是否足够进入可表达域，从而把推理从“写更多”转为“结构风险下降”。

它不是万能指标。

需要明确：E 不是正确性证明，也不等价于“信息充分”。它是一种工程可操作的风险代理：在你接受的结构假设与验证器约束下，E 越低意味着“更不容易在表达阶段出现典型失败模式”。不同领域需要不同的冲突判定规则、不同的证据条件与不同的权重配置，这是开放问题，也是工程落地的必经步骤。

参数极端化下的系统行为

当 $\varepsilon \to 0$，系统极端保守，S2 更深但更稳健；当 $\varepsilon \to 1$，系统趋近传统生成范式，S2 很少触发但结构风险上升。α, β, γ 的极端配置对应不同风险敏感性与路径依赖程度。

除了 ε，还有三个常被忽视的旋钮。

- k**（S1 检索规模）：**太小会导致覆盖不足或错过冲突对；太大则增加噪声与冲突，抬高 E_{conf} 并拖累稳定性。
- $T_{\max}$**（S2 预算）：**太小会导致“本可调解却提前失败”；太大则增加成本，且可能在弱约束下产生过多合成结构。
- **验证器阈值：**证据相似度阈值太低会放过 bridge-only 过门；太高则导致过度拒答。

这些旋钮共同决定系统的“保守程度、成本上限与证据严格性”，建议在固定任务集上做系统化对比实验来得到稳定结论。

与现有范式的关系

本文框架可作为认知约束层与 Transformer、RAG、执行模块协同：它不替代生成器，而约束“什么时候允许表达”，并提供结构推理与风险控制接口。

与 RAG 的关键差异。

RAG 通常把检索结果直接喂给生成器；本文则要求检索结果先通过熵判别与结构重构：如果内部冲突高或证据不足，即便检索到了相关片段也不允许直接表达。换言之，本文把“检索到信息”与“允许表达”显式解耦。

如何与 LLM 协同。

LLM 可以参与两类位置：一是作为表达层 $G(q, M)$ 的投影器；二是作为 S2 的候选生成器（提出可能的 rewrite）。但无论 LLM 多强，本文要求它**不直接决定过门**：任何候选结构都必须满足 No-DROP 与严格降熵，并通过外部验证器的证据条件。这把 LLM 的强生成能力纳入一个可审计的风险控制框架里。

逻辑的“非显式性”与可解释性问题

本文不预设形式逻辑语法，牺牲部分“逻辑可读性”，换取更通用的结构表达载体。可解释性更应体现为“为什么某种结构变化降低了熵”，而非“是否符合某套预设逻辑语法”。

解释的粒度来自 trace。

虽然不使用显式逻辑推演，但系统仍然可以提供工程可用的解释：每一步 S2 新增了什么结构、哪一项熵下降了、为什么验证器通过或失败。相比“给出一段思维链文本”，这种解释更容易被自动检查与回放，也更适合做调参和回归测试。

潜在局限与开放问题

开放问题包括：熵函数与冲突判定的任务依赖性、REWRITE 搜索空间的规模控制、虚拟环境构造的抽象性与泛化等。

额外两点工程风险。

- **冲突判定的可靠性：**若 χ 过于粗糙，可能产生“假冲突”导致过度拒答；过于宽松又会放过矛盾结构导致不安全表达。
- **基准任务的可迁移性：**合成任务能验证闭环，但仍需更多真实场景任务来检验“熵门控是否真的降低实际事故率”。

这些问题并不否定框架，而是指出下一步工程化必须解决的验证与数据问题。

结论（Conclusion）

本文提出一种以可表达性熵为核心判据、以 REWRITE 为动力的认知模型框架：系统在状态 (q, M) 上推理，答案文本被排除在状态之外；通过熵判别器决定是否允许进入表达；若未通过，则 S2 在单调扩展与严格降熵约束下新增结构直至收敛或终止。进一步地，引入虚拟环境训练以沉淀有效降熵模式，并以记忆熵守恒约束保证长期容量稳定。整体上，框架把“是否该回答”变成一个可计算、可调、可验证的工程对象。

一句话总结。

推理的本质不在于把文本写出来，而在于让系统状态变得**可以被稳定表达**；语言生成只是最后一步投影，而不是推理的驱动力。

下一步工作。

若要进一步走向工程应用，关键在于建立更丰富的真实任务集与更严格的外部验证器设计，把“风险降低”与“事故减少”建立可量化关联；并在更复杂记忆结构（图、约束网络、多模态证据）下探索冲突判定与容量守恒策略。