轨迹知情的自我改进智能体系统记忆生成

Gaodan Fang、Vatche Isahagian、K. R. Jayaram、Ritesh Kumar、Vinod Muthusamy、Punleuk Oum、Gegi Thomas*Agents and AutomationLab，IBMResearch 美国

arXiv:2603.10600v1 [cs.AI] 11 Mar 2026

摘要

大语言模型驱动的智能体面临一项持续挑战：从自身的执行经验中学习，以提升未来表现。尽管智能体能够成功完成许多任务，但它们往往重复低效模式，无法从类似错误中恢复，并错失将以往执行中的成功策略加以应用的机会。我们提出一种新颖框架，可从智能体执行轨迹中自动提取可操作的经验，并通过上下文记忆检索利用这些经验来提升未来表现。我们的方法包含四个组件：（1）轨迹智能提取器，对智能体推理模式进行语义分析；（2）决策归因分析器，识别哪些决策和推理步骤导致了失败、恢复或低效；（3）上下文学习生成器，生成三类指导——来自成功模式的策略提示、来自故障处理的恢复提示，以及来自低效但成功执行的优化提示；（4）自适应记忆检索系统，基于多维相似度将相关经验注入智能体提示词。与存储通用对话事实的现有记忆系统不同，我们的框架理解执行模式，提取带有来源追溯信息的结构化经验，并检索针对特定任务上下文量身定制的指导。在 AppWorld 基准上的评估表明改进持续稳定，在留出任务上的场景目标完成度最高提升 14.3 个百分点，对复杂任务的收益尤为显著（场景目标提升 28.5 个百分点，相对增幅达 149 %）。

CCS概念

•**计算方法论** → **信息抽取**; **多智能体系统**; **知识表示与推理**; •**信息系统** → 企业应用; 信息检索.

关键词

智能体记忆，自我进化智能体

ACM参考文献格式：

GaodanFang、Vatche Isahagian、K. R. Jayaram、Ritesh Kumar、Vinod Muthusamy、Punleuk Oum、Gegi Thomas。2026。轨迹知情的自我改进智能体系统记忆生成。载于描述研究的技术报告

*作者姓名按字母顺序排列。

允许为个人或课堂使用而免费制作本作品全部或部分内容的数字副本或纸质副本，但不得以营利或商业利益为目的复制或传播，且副本首页必须附有本声明及完整引文。必须尊重本作品中第三方组件的版权。其他用途请联系所有者/作者。技术报告，纽约州约克镇高地© 2026 版权归所有者/作者所有。

https://doi.org/10.1145/nnnnnnn.nnnnnnn

由 *IBM* 完成。ACM，纽约州纽约，14 页。https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 引言

大语言模型（LLM）驱动的智能体已能够越来越复杂地自动执行各种任务，范围涵盖网页导航到 API 编排。这些智能体通过反复推理任务、选择操作、执行操作并观察结果来运行。然而，一个根本性的局限依然存在：智能体会失忆，因为大多数大语言模型都是无状态的。智能体缺乏从执行经验中系统学习的机制 [4, 17]。如果某个智能体今天在特定的 API 身份验证流程上遇到困难，明天仍会在相同流程上举步维艰，除非手动更新其提示词。智能体发现某项任务的高效策略后，无法自动将该策略应用于未来类似的任务。智能体成功从错误中恢复，也无法让未来遇到类似错误的执行过程受益。

考虑一个简单的电子商务任务：将商品加入购物车并完成结账。智能体可能成功完成该任务，但效率低下——例如，循环调用 amazon_remove_from_cart(item_id) 来清空购物车，而一次 amazon_empty_cart() 调用即可完成。在另一次执行中，智能体可能因未先添加支付方式就尝试结账而完全失败，随后通过识别错误并添加支付信息成功恢复。在又一次执行中，智能体可能从一开始就干净利落地执行任务，在每次操作前系统性地验证前置条件。

这些轨迹中的每一条都包含对未来执行有价值的经验，但类型各不相同。低效的成功提示了一条**优化提示**：清空含有多件商品的购物车时，应使用批量操作，而不是逐一移除。先失败后恢复提示了一条**恢复建议**：当结账因缺少支付方式而失败时，应在重试前确认支付信息已配置。干净利落的成功提示了一条**策略提示**：在启动结账操作之前，应系统性地验证所有前置条件，包括购物车内容、收货地址以及支付方式是否可用。

当前的智能体改进方法不足以捕捉这些多样化的学习机会。基于规则的系统要求开发人员手动预判模式，并将其编码为决策规则，因此十分脆弱，无法适应意料之外的情况。提示工程通过迭代优化指令和示例来改进常见模式，但这类指导较为通用，并不针对实际部署经验，也没有根据观察到的结果自动改进的机制。通用记忆系统 [2, 15]存储

从对话中提取的事实，并存入向量数据库以供日后检索，但这类系统缺少若干关键能力：它们不了解智能体执行模式和推理流，无法进行因果分析以确定哪些决策导致了失败或效率低下，也缺乏按策略、恢复和优化等类别进行结构化学习提取的能力，并且无法追溯学习内容的来源轨迹。近期研究已开始从智能体轨迹中提取可复用知识，包括成功执行的工作流 [6, 13], 、程序性指令 [5], 、推理策略 [9],以及演化上下文剧本 [16]，但这些方法通常只从成功轨迹中学习，缺乏对失败的明确因果归因，或是生成整体式文档，而非结构化、可检索的记忆条目。进一步的实证研究表明，简单累积经验会导致错误传播和错位回放 [14], ，这凸显了开展质量感知记忆策展的必要性。

我们提出了一种框架，通过轨迹引导的记忆生成与检索来解决这些局限。我们的核心洞见是，智能体的执行历史——即轨迹——包含丰富的语义信息，不仅揭示了发生了什么，还说明了智能体为何做出决策、如何推理任务、哪些策略奏效、哪些模式效率低下，以及决策链如何导致失败与恢复。通过对这些轨迹进行语义理解，我们能够自动提取多个类别的可操作经验，将失败和低效归因于具体决策与推理步骤，生成上下文感知指导，并依据多个上下文维度检索相关经验。

我们的贡献如下：

- 我们提出了轨迹智能提取方法，超越原始日志记录，实现对智能体推理模式的语义理解，包括分析性思考、规划模式、验证行为、反思模式和自我纠正序列。
- 我们提出了自动决策归因方法，可区分失败的直接原因、近因和根本原因，同时识别哪些决策促成了成功恢复，以及哪些执行模式虽取得成功却效率低下。
- 我们开发了上下文学习生成方法，可生成三种不同类型的指导：策略提示，用于编码顺利成功的执行中体现的成功模式；恢复提示，用于捕捉故障处理和错误纠正方法；优化提示，用于从成功但并非最优的执行中识别效率改进机会。
- 我们设计了自适应记忆检索机制，将语义相似度与元数据过滤及基于优先级的排序相结合，确保智能体能够根据特定情境（包括任务类型、领域和执行模式）获取最相关的指导。
- 我们在 App-World 基准测试中展示了该框架的有效性：它在所有难度等级上均取得了稳定提升，尤其是在复杂任务上表现突出，而这些任务正是经验最有价值的场景。

2 问题陈述

2.1 智能体学习挑战

大语言模型驱动的智能体通过反复推理、选择操作并观察结果来执行任务。每条执行轨迹——从初始请求到最终结果的完整思考、操作和结果序列——都包含可为未来执行提供参考的模式 [11]。然而，由于种种原因，从这些轨迹中提取可付诸行动的经验并非易事。

首先，**不同结果类别中都存在有价值的模式**。并非所有学习机会都来自失败。成功完成任务的智能体可能采用了值得效仿的优雅策略，发现了高效的 API 使用模式，或执行了全面的验证序列，从而避免了错误。相反，最终成功的智能体也可能效率低下——采取不必要的步骤、重复调用 API，或在已有批量操作时使用细粒度操作。而遭遇失败的智能体也可能成功恢复，其恢复模式本身就值得记录。全面的学习系统必须从顺利成功、低效成功、先失败后恢复序列以及彻底失败中提取经验。

其次，**因果关系往往无法从原始日志中直接看出**。当智能体在执行的第 15 步失败时，导致问题的决策可能早在第 3 步就已做出。当智能体成功从错误中恢复时，要确定哪些具体推理促成了恢复，需要对智能体的思考进行语义理解，而不能只观察最终结果。当智能体低效地完成任务时，要判断哪种替代方法更高效，需要同时理解智能体做了什么，以及还有哪些其他选项。

第三，**必须根据上下文检索经验**。关于使用批量购物车操作的优化建议，在智能体执行购物车管理时相关，但对于撰写电子邮件任务则无关。关于如何处理身份验证失败的恢复建议，对于涉及经过身份验证的 API 的任务至关重要，但对于只读操作则没有必要。检索系统必须根据多个维度，将经验与上下文相匹配：任务类型、领域、与当前请求的语义相似度，以及涉及的具体执行模式。实证证据表明，智能体会严格遵循检索到的经验 [14], ，这进一步凸显了精准检索的重要性，也意味着检索不匹配会直接导致性能下降。

第四，**经验必须可执行且具体**。诸如“对API调用要小心”之类的笼统建议价值不大。有效的经验会明确具体的验证检查、特定的API 使用模式、具体的错误恢复序列，或明确的前置条件验证步骤。它们必须以智能体能够直接应用的方式来表述：“在开始结账之前，通过调用 get_payment_methods() 并检查结果是否非空，验证支付方式已配置”是可执行的；“确保支付可用”则不是。

第五，**学习成果必须能够追溯到其来源**。每项学习成果都必须保留来源追溯信息——链接回其所源自的特定轨迹和结果 [3]。这样便能验证学习成果是否有效（部署学习成果后，类似的失败是否仍会发生

？）、调查生成特定指导的原因，并审计学习系统的决策。缺少来源追溯信息，就无法调试错误的指导、评估学习质量随时间的变化，也无法建立对系统建议的信任。

2.2 学习要求

对于以迭代方式进行推理和行动的智能体（例如，ReAct 风格、规划与执行），学习系统必须满足若干要求。

从成功模式中提取策略：当智能体顺利执行任务——没有错误、不必要的步骤或恢复流程——其处理方式往往蕴含着有效策略。系统必须识别这些模式：智能体是否在尝试操作前验证了先决条件？是否在选择 API 前系统地探索了可用 API？是否在继续执行依赖步骤前验证了中间结果？这些成功模式应编码为策略提示，引导未来的执行采用同样有效的方法。

从故障处理中提取恢复经验：当智能体遇到错误但成功恢复时，恢复流程具有重要价值。系统必须识别哪里出了问题、智能体对故障有哪些认识、它如何调整处理方式，以及哪些具体操作促成了成功恢复。例如，如果智能体在未配置支付方式的情况下尝试结账，收到错误后意识到缺少支付方式，补充支付信息并成功重试，那么整个流程都应编码为恢复建议，其中包括失败模式、识别信号和纠正步骤。

从低效成功中提取优化经验：当智能体成功完成任务，但做法并非最优时，系统必须识别其中的低效之处，并找出更高效的替代方案。这不仅需要了解智能体做了什么，还需要弄清楚当时有哪些其他选项。例如，如果存在批量清空_购物车() 操作，智能体却在循环中逐项移除购物车中的商品，系统就必须识别这种模式，找出更高效的替代方案，并编写优化提示，说明何时以及如何使用批量操作。

步骤级决策归因：当出现失败或低效情况时，系统必须识别导致相应结果的具体推理步骤和决策。这需要对智能体的思考进行语义分析，而不仅仅是观察其行动。如果智能体因未核实某个 API 是否可用便认定它可用而失败，归因就必须指出这一假设步骤，解释其问题所在，并说明本应进行哪些核实。

思维模式识别：智能体常常表现出元认知 e
行为，这些行为体现了其推理质量。明确核实前置条件的智能体表现出积极模式。能够识别自身错误并自我纠正的智能体体现了反思。未经核实便作出假设的智能体则表现出消极模式。系统必须从语义层面识别这些认知模式——例如，即使没有使用“验证”一词，“我应该核实所有 API 是否可用”也体现了验证模式——并利用这些模式指导学习提取。

语义推理分析：除了识别单独的思考模式外，系统还必须超越原始执行日志，理解智能体推理的完整结构。系统必须识别并分类不同的推理模式——分析性思考（检查数据或约束）、规划性思考（制定行动序列）、验证性思考（检查前置条件或中间结果）、反思性思考（评估过去的行动）以及自我纠正序列（识别错误并从中恢复）——从而理解智能体如何对任务进行推理，以及其推理在哪些方面取得成功或失败。这种对推理流的结构化理解，使系统能够从轨迹中提取有意义的学习成果，而不是仅仅对行动进行表层的模式匹配。

2.3 现有方法的局限性

现有的智能体改进方法未能全面应对这些挑战。

基于规则的系统 依据预期模式编码决策规则，但无法适应未曾预料的情况；随着新模式出现，还需要持续进行人工维护。它们也无法从观察到的执行轨迹中自动提取规则——每条规则都必须由开发人员手动编写，而开发人员可能无法了解实际部署模式。

提示工程 通过反复完善指导和示例来提升智能体性能，但这些指导内容较为通用，并非针对实际部署经验。如果智能体在某个特定的 API 身份验证流程中屡次失败，提示工程最终或许能捕捉到这一模式，但这需要先经过人工观察并修改提示词。该方法没有基于观察到的结果自动改进的机制，也没有系统化的方法从成功、失败和恢复中全面捕捉学习机会。

通用记忆系统 是一种更复杂的方法，但仍有不足。Mem0 [2] 和 Letta [10] 等系统会将从对话中提取的事实存储在向量数据库中，以供日后检索。然而，这些系统缺少智能体学习所需的几项关键能力。它们不了解智能体执行模式——不会区分策略模式、恢复序列和优化机会，而是将所有记忆一视同仁。它们无法进行因果分析，以识别哪些决策导致了失败或效率低下——它们存储结果，却不记录产生这些结果的决策链。它们缺乏包含类别、优先级和可执行步骤的结构化学习提取机制——记忆通常是自由格式的文本，不具备为智能体提供指导所需的结构。它们也无法将学习内容溯源至源轨迹，因此无法验证学习是否有效，也无法调查为何生成某些指导内容 [17]。

强化学习方法从奖励信号中学习，但针对这一问题有其自身局限。它们需要大量训练数据才能学习有效策略，而当失败虽罕见却后果严重时，这些数据可能无法获得。它们的训练和更新计算成本高昂，因此不适用于持续演进的智能体系统。它们对某些决策为何能改善结果的可解释性有限——所学策略往往是黑箱。对于理解改进背后的推理具有价值的场景（例如调试或审计智能体行为），强化学习方法提供的透明度不足。此外，强化学习方法难以应对

多类别学习问题——它们针对总体奖励进行优化，但并不能自然地区分策略模式、恢复序列和优化机会。

3 方法

如图1所示，我们提出一个框架，将原始智能体执行轨迹转化为可操作的、按上下文检索的指导，供未来调用使用。该框架以三阶段流程运行：

(1) **阶段1：轨迹分析与提示提取**。 给定智能体针对已完成任务的执行轨迹，系统分析推理轨迹以识别因果决策链——即结果为何发生——并提取结构化提示，以捕捉有效策略、恢复模式和优化机会。提示在两种互补粒度上提取：*任务级* 提示捕捉整体端到端模式，以及 *子任务级* 提示将轨迹分解为可复用的逻辑阶段（身份验证、数据检索、处理等），以实现跨任务迁移。

(2) **阶段2：提示存储与管理**。 提取的提示在存储前会经过泛化、聚类和整合。子任务描述被抽象化以去除特定实体细节，从而对来自不同任务、但共享共同子任务模式的提示进行语义聚类。基于大语言模型的合并流程会整合每个聚类内冗余或重叠的提示，形成经过筛选的、无冗余的高质量指导记忆。提示以双重表示存储——用于语义搜索的向量嵌入，以及用于过滤的结构化元数据。

(3) **阶段 3：运行时检索**。 当智能体接到新任务时，系统会从记忆中检索相关提示，并在推理开始前将其作为指南注入智能体的提示词中。系统支持两种检索策略：余弦相似度检索（速度快，无需调用大语言模型）和大语言模型引导的选择（能够对任务上下文进行更深入的推理，但需要额外调用一次大语言模型）。

这些阶段构成了一个自强化循环：随着处理的轨迹越来越多，记忆系统会积累日益全面、完善的指导。获得这些指导的智能体能够生成质量更高的轨迹，而这些轨迹可能揭示更细微的模式，以供进一步学习。以下各小节将详细介绍每个阶段。

3.1 阶段1：轨迹分析与提示提取

此阶段会分析已完成的智能体轨迹，以提取结构化、可执行的提示。它包含三个阶段：轨迹信息提取、决策归因分析和提示生成。提示生成阶段的一个关键设计维度是提取提示的粒度——既可以针对完整任务轨迹（任务级），也可以针对轨迹中的各个逻辑子任务（子任务级）。我们将在第4节中探讨这两种粒度并比较其效果。

3.1.1 轨迹智能提取器。 轨迹智能提取器将原始智能体执行数据转换为结构化的中间表示，所捕捉的语义含义超越传统日志记录所提供的内容：智能体为何做出特定决策、如何验证其推理、在何处表现出自我纠正行为，以及哪些模式刻画了成功执行与不成功执行。

该组件接收原始智能体轨迹，其中包含按顺序排列的步骤，包括智能体调用、提示词或上下文、包含思考和反思的智能体响应、采取的行动及其结果，以及可选的评估报告或真实结果评估。每条轨迹都代表一次完整的任务执行，从用户最初的请求到最终结果。至关重要的是，系统不要求提供真实结果标签（成功或失败）：如果有此类标签（例如来自基准评估框架），系统便直接据此对轨迹进行分类；如果没有，则根据后续阶段识别出的智能体自身的自我反思信号来推断结果。

第一个处理阶段会解析智能体响应，并根据认知功能将推理识别并归为四种类型：**分析性思考**，即智能体分析情况并评估约束；**规划性思考**，即智能体决定采取哪些行动以及行动顺序；**验证性思考**，即智能体检查假设或验证前置条件；以及**反思性思考**，即智能体重新审视其方法，这通常由意外结果触发。除了分类之外，提取器还会识别状态指标、执行摘要和错误识别陈述，使下游组件能够理解促成这些行动的推理过程。

第二阶段通过语义理解而非关键词匹配，使用大语言模型识别提取的思考中的认知模式。系统能够识别：**验证模式**——任何检查或验证假设的表达，即使其中不包含与验证相关的关键词（例如，“我需要确保所有必需的 API 都已包含”就体现了验证行为）；**反思模式**——重新考虑之前的决策，通常发生在错误之后；**自我纠正模式**——在外部信号出现之前主动识别并修正错误；**错误识别模式**——注意到可能影响任务完成的问题；**API 发现模式**——系统地探索可用的 API；以及**效率意识模式**——考虑是否存在更高效的替代方案。这种语义方法能够跨越语言表达差异进行泛化，不同于基于规则的关键词匹配。

第三阶段确定轨迹结果。当存在真实标签评估报告时，该阶段通过语义理解对其进行解释：报告中“API 响应返回 400 错误请求”会被转换为“结账 API 失败，因为未提供必需的支付方式参数”，并且针对每个结果指标，模块会确定测试验证了什么、失败的原因（如适用）、对任务完成的影响以及总体质量评估。当缺少真实标签时，该阶段则根据第 1 阶段和第 2 阶段提取的自我反思信号——反思性思考、自我纠正模式和错误识别模式——综合生成结果，以推断智能体是成功、失败还是恢复。在这两种情况下，最终都会得到供下游组件使用的结果分类。

第四个阶段专门分析成功执行，并区分：**顺利成功**——任务完成时没有发生错误或执行不必要的步骤，其中的模式可作为策略建议的候选；**低效成功**——任务已完成，但未达到最优效果（例如，

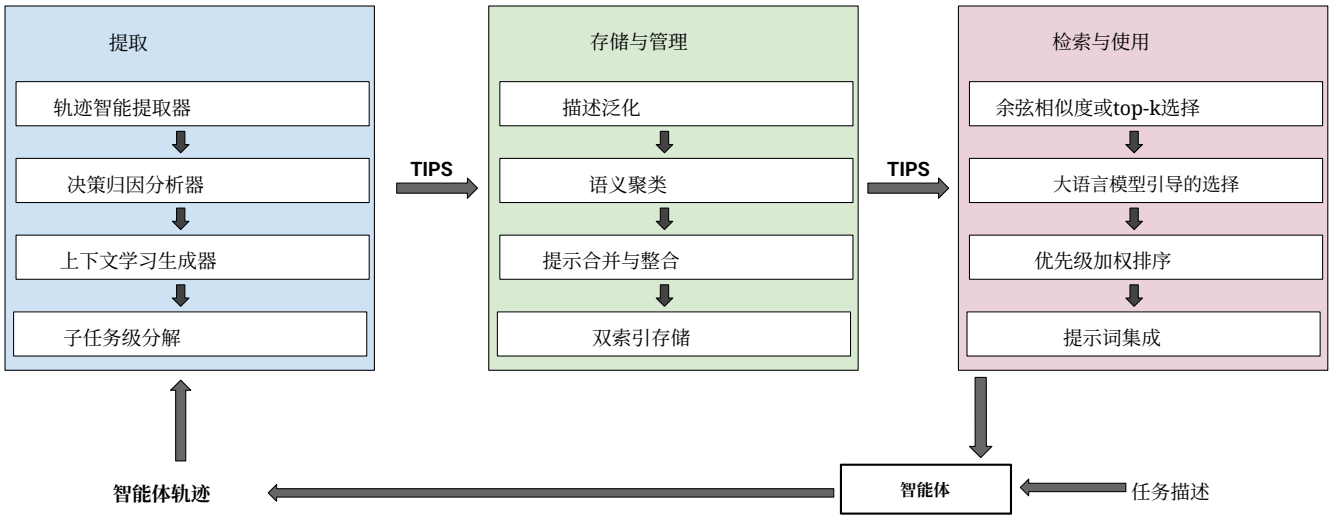

图1：我们方法的概述

可以合并执行却重复操作），由此得出的候选内容可作为优化提示；以及**恢复序列**——在其他方面成功的执行过程中，成功处理错误，由此得出的候选内容可作为恢复提示。

输出是一种结构化中间表示，其中丰富了提取的思考、带有置信度分数的识别出的模式、评估信息、成功模式，以及包括轨迹标识符、任务意图、步骤数和总体结果分类在内的元数据。

3.1.2 决策归因分析器。 决策归因分析器执行自动化因果分析，以识别哪些决策和推理步骤导致了所观察到的结果。它分析所有结果类型——而不仅仅是失败。

第一阶段扫描中间表示，以查找四个类别的结果指标：**失败指标**——失败的评估、错误消息、任务未完成信号；**恢复指标**——失败后成功完成、识别错误后采取纠正措施；**低效指标**——本可批量处理的重复操作、不必要的中间步骤、存在批量替代方案的细粒度操作；以及**成功模式**——干净完成、系统性前置条件验证、高效的 API 使用。对于每个检测到的结果，都会提取上下文信息，作为因果分析的起点。重要的是，结果所在位置通常并非原因所在位置。

因果分析模块使用大语言模型，沿着智能体的推理步骤向回追溯，以确定哪些决策导致了观察到的结果。对于**失败**，分析会区分：直接原因（直接触发失败的因素）、近因（促成失败的近期决策）、根本原因（引发整个链条的底层问题）以及促成因素。对于**恢复**，分析会确定是什么促成了失败、智能体如何识别问题、采取了什么纠正措施，以及纠正为何能够成功。对于**低效**，分析会确定是什么导致执行未达到最优、有哪些更高效的替代方案、替代方案为何更好，以及智能体是否意识到了这种低效。对于**成功模式**，分析会确定哪些策略促成了顺利成功、这些策略为何有效，以及是什么使该方法尤为出色。

最后阶段会针对每个归因决策点生成具体的预防或改进措施。这些措施必须**可执行**——智能体能够实际采取这些措施；必须**具体**——提出明确的行动，而非模糊的建议；必须**针对原因**——直接解决根本原因；并且必须**具有预防或改进作用**——防止类似失败再次发生，或明确指出更高效的做法。

3.1.3 上下文学习生成器。上下文学习生成器会将决策分析转化为可重复使用的记忆条目，确保这些条目可执行、富含上下文信息且分类恰当。其关键创新在于根据轨迹结果生成三种不同类型的提示。

策略提示从顺利成功的执行中提炼有效模式——哪些做法行之有效，值得重复采用。例如：

> 内容："执行结账操作时，应在启动结账流程前，系统地核实所有前置条件（购物车内容、收货地址、支付方式）。"类别：策略 步骤：。

[1] 调用 get_cart_items() 以确认购物车不为空 2. 调用 get_shipping_address() 以确认地址已配置 3. 调用 get_payment_methods() 以确认存在支付方式 4. 仅在满足所有前置条件后继续结账 触发条件："当任务涉及结账、购买或支付操作时"

恢复提示 从先失败后恢复序列中同时编码失败模式和恢复模式。示例：

> 内容："当结账因 '需要支付方式 '错误而失败时，请检查支付配置；如果缺少支付方式，请先添加，再重试。"类别：恢复 步骤：1. 识别表明缺少支付方式的错误消息 2. 调用 get_payment_methods() 检查当前配置 3. 如果配置为空，则使用适当的详细信息调用 add_payment_method() 4. 重试结账操作 触发条件："当结账或支付操作失败时" 反例："不要在未处理缺少支付方式的问题时直接重试。"

优化提示 从成功但不够理想的执行中找出效率改进之处。示例：

> 内容："清空含有多个商品的购物车时，请使用 empty_cart()，而不是逐个商品循环调用 remove_from_cart(item_id)。"类别：优化 步骤：1. 检查购物车中是否有多个待移除商品 2. 不要循环调用 remove_from_cart()，而是调用一次 empty_cart() 3. 使用 get_cart_items() 确认购物车为空 触发条件："当任务要求移除购物车中的所有商品时" 反例："清空整个购物车时，不要使用 for i in items: remove_from_cart(i)。"

系统会分析轨迹，以确定生成和检索所需的上下文维度：应用上下文（任务涉及哪个领域）、任务类别 （该领域内的操作类型）以及复杂度级别。系统会针对每个类别使用专用提示来生成提示内容，并纳入相关的执行模式；每条提示词都包含指南，用于生成可执行、具体且可泛化的提示。

每个生成的记忆条目都包含：唯一标识符、提示类别（策略、恢复、优化）、可执行内容、解释目的、具体实施步骤、触发条件、可选反例、应用上下文（通用提示则为 null）、任务类别（通用提示则为 null）、优先级（根据结果严重程度分为 critical/high/medium/low）、源轨迹 ID，以及源结果描述。

该系统还会从同一条轨迹中同时生成领域专用提示和通用提示，从而最大限度地提高精确度和覆盖范围。针对电子商务结账中缺少支付 API 的失败案例，系统会生成一条领域专用提示（"对于涉及结账的电子商务任务，请在启动结账前确认支付方式已配置"）和一条通用提示（"在启动具有前置条件的操作时，请在开始前系统地核实所有前置条件"）。这种双层泛化机制可确保：当上下文与领域专用提示匹配时具有较高的精确度，同时通过适用于新颖领域的通用提示实现广泛覆盖。

3.1.4 任务级与子任务级提取。 提示生成阶段可以在两种粒度下运行。任务级 提取将整条轨迹视为一个整体，生成能够捕捉端到端执行模式的全面提示。子任务级 提取则首先将轨迹分解为逻辑子任务，然后分别为每个子任务提取有针对性的提示。

这两种方法各有利弊。任务级提示易于提取，能够涵盖贯穿整个任务的总体策略。然而，其可复用性受任务具体性的限制：从"说出 Spotify 上最常向我推荐的艺人"中提取的提示，可能无法迁移到"将我常用的入睡闹钟延后 20 分钟"，尽管两者都包含身份验证和分页数据检索等常见子任务。任务级提示还会将不同执行阶段的事项捆绑在一起，降低检索精度。

子任务级抽取通过将每条提示限定于单一逻辑阶段来解决这些局限性。许多任务共享可跨上下文泛化的常见子任务：

- **身份验证子任务**在各应用（Spotify、Phone、Venmo）中遵循共同模式：从监督者处检索凭据、登录并存储访问令牌。
- **数据检索子任务** 共享分页模式：发起分页API调用、汇总结果，并将其存储以供下游处理。
- **数据处理子任务** 涉及与领域无关的操作：对检索到的数据进行计数、过滤、汇总和转换。
- **任务完成子任务** 几乎普遍适用：报告结果并将任务标记为完成。

通过以这一粒度提取提示，我们实现了跨任务迁移（来自Spotify任务的身份验证提示有助于Phone应用任务）、更好的匹配（关于更新闹钟的任务会从"检索所有闹钟"子任务中检索提示，即使原始任务是关于删除闹钟），以及组合学习（新的复杂任务可利用来自多个更简单子任务的提示）。

两阶段抽取流水线。 子任务级提取以基于大语言模型的两阶段流水线运行。

阶段A：轨迹分割。 大语言模型会分析完整的智能体轨迹，并将其分割为逻辑子任务。对于每个子任务，模型会生成一条泛化描述（有意保持通用，例如"通过 Spotify 进行身份验证"，而不是"以 user@gmail.com 用户身份登录"）、所涉及的应用程序集合、该子任务在原始轨迹中的步骤范围（以保持可追溯性），以及该子任务的目的。分割提示会指导模型识别不同逻辑阶段之间的自然边界——例如从身份验证过渡到数据检索、从数据检索过渡到处理，依此类推。

例如，对于"说出 Spotify 上向我推荐最多的艺术家"这一任务，其轨迹可能会被分割为：(1) 查找相关 API 及其规范；(2) 通过 Spotify 进行身份验证；(3) 通过分页请求检索推荐歌曲；(4) 分析推荐结果，以确定被推荐最多的艺术家。

阶段B：子任务级提示提取。 大语言模型会针对每个子任务独立提取 2–4 条可执行提示。将每次提取调用限定在单个子任务范围内，可使提示词保持聚焦，避免提示混淆不同执行阶段的关注点。这些

提示必须具体（提供特定的 API模式，而非含糊的建议）、具有普适性（避免电子邮件地址、歌曲名称或付款金额等特定任务细节），且可执行（智能体遇到类似子任务时可直接应用）。也可以在阶段A和阶段B使用不同的模型——使用能力更强的模型进行分段，使用更轻量的模型进行子任务级提取——以平衡成本与质量。

“使用 Spotify 进行身份验证”子任务的示例输出：

```
提示：。
1
    ”尝试身份验证前，务必从 supervisor.show_account_
passwords() 检索账户凭据” 2. ”登录后立即存储并验证访问令
牌，以确保后续 API 调用成功” 3. ”按应用名称筛选凭据，为目
标服务选择正确的密码”
```

子任务级提示与任务级提示相辅相成，而非相互竞争。任务级提示总结端到端执行策略的整体模式（例如，“结账前确认所有前置条件”），而子任务级提示则总结特定执行阶段的重点模式（例如，“获取大量结果集时使用分页检索”）。这两个层级的提示都存储在同一记忆系统中，并可在阶段3期间一并检索。

3.2 阶段2：提示存储与管理

随着提示从涵盖多样任务的众多轨迹中不断积累，记忆系统必须应对冗余、不一致与可扩展性问题。两条涉及电子商务结账的轨迹可能各自独立地产生关于核验支付方式的提示；跨越不同应用的数十条轨迹则会产生产生指导内容相互重叠的身份验证相关提示。若不加以巩固，记忆会随已处理轨迹的数量线性增长，近乎重复的提示会争夺有限的提示词空间，从而降低检索质量，而来自不同轨迹的相互矛盾的提示也可能使智能体感到困惑。

阶段2通过由子任务描述泛化、语义聚类和基于大语言模型的提示整合构成的流程来应对这些挑战。

3.2.1 子任务描述泛化。 阶段1生成的子任务描述具有不同程度的具体性，从而妨碍聚类。“使用监督者API检索john.doe@email.com的Spotify密码”、“获取用户alice_smith的Venmo登录凭据”以及“从监督者处获取Phone应用密码”，描述的都是同一抽象操作：检索服务凭据。为了实现有意义的聚类，系统通过三种变换对子任务描述进行泛化：

- **实体抽象**：将特定用户名、电子邮件地址、应用名称、项目ID及其他实体引用替换为通用占位符。“为 john.doe@email.com 获取 Spotify 密码”会变成“获取服务账户凭据”。
- **动作规范化**：将语义等同的动词和短语映射为规范形式。“Get”“fetch”“retrieve”和“obtain”会统一为一个规范动词。“Log in”“sign in”和“authenticate”也会以同样的方式统一。
- **上下文移除**：删除不会影响子任务核心操作的任务特定上下文限定语。“为检查订阅状态而检索凭据”会简化为“检索服务账户凭据”，因为后续用途不会改变凭据检索的执行方式。

这些转换通过大语言模型完成，提示词会指示模型在保留核心操作的同时，生成尽可能抽象的描述。泛化描述是聚类的基础：泛化子任务描述在语义上相似的提示，可能包含重叠或互补的指导。

3.2.2 语义聚类。系统通过计算各提示的向量嵌入之间的余弦相似度对提示进行聚类，然后应用带有相似度阈值的层次凝聚聚类。诸如“获取服务账户凭据”和“通过外部服务进行身份验证”这样的泛化描述，尽管表面上相关，却可能描述不同的子任务；而“获取服务账户凭据”和“获取应用登录凭据”描述的则是同一操作。使用适当阈值的层次聚类（根据经验，泛化描述的阈值为 ~0.85）会将真正等价的子任务描述归为一组，同时将不同操作区分开来。

在每个簇中，都会收集所有相关提示，而不论其来源轨迹、任务上下文或提取粒度如何。“检索服务账户凭据”簇中可能包含来自 Spotify 身份验证轨迹、Venmo 登录轨迹和 Phone应用凭据检索的提示——这些提示都反映了在不同任务中观察到的同一种底层子任务模式。

3.2.3 提示整合与合并。在每个簇中，基于大语言模型的巩固流程会合并冗余提示、解决冲突，并生成一组经过筛选且互不重叠的指导。巩固分为三个步骤：

去重。识别并合并内容几乎相同的提示。“登录前始终先调用 show_account_passwords()”和“身份验证前使用主管密码 API 检索凭据”传达了相同的指导；巩固流程会生成一条规范提示，概括二者共有的见解。

冲突解决。当不同轨迹中的提示给出相互矛盾的指导时（例如，一条提示建议身份验证失败后立即重试，另一条则建议先重新检索凭据），系统会利用结果元数据——提示类别、优先级以及来源轨迹的成功/失败状态——来判断哪条指导更可靠。来自成功轨迹的提示优先于来自失败轨迹的提示；编码了经验证的纠正模式的恢复提示，则优先于推测性的预防策略。

综合。针对同一子任务不同方面的互补提示会综合成连贯、全面的指导。例如，一条提示介绍凭据检索，另一条提示介绍登录后的令牌验证，那么整合后的输出会将两者合并为一条提示，按顺序列出各个步骤，涵盖完整的身份验证工作流。

巩固过程还会生成规范化的簇描述——一条概括性的子任务描述，用于表示该簇以供检索。该描述会重新嵌入，并与整合后的提示一同存储，取代各条轨迹原有的描述。

3.2.4 存储表示。 每条整合后的记忆条目都采用两种互补的表示形式进行存储。**向量嵌入**是使用文本嵌入模型根据提示内容和目的计算得出的稠密向量。它能够捕捉语义含义，从而支持跨不同术语的相似性搜索。例如，关于“续订订阅”的提示可以匹配提及“延长我的会员资格”的任务描述；关于“安排定期活动”的提示可以匹配“设置每周会议”。**结构化元数据**由一组可筛选属性构成：提示类别（策略、恢复、优化）、优先级、应用上下文、任务类别、源轨迹 ID（复数形式，因为整合后的提示可能源自多条轨迹）以及创建时间戳。

子任务级提示按其规范簇描述建立索引，任务级提示则按原始任务描述建立索引，从而形成自然分组，支持在这两个粒度层级上进行检索。

3.3 阶段 3：运行时检索

当智能体被调用来执行一项新任务，且任务描述为d时，系统会从记忆中检索相关提示，并在推理开始前将其作为指南注入智能体的提示词中。检索策略会直接影响智能体收到的是相关、可执行的指导，还是被无关提示分散注意力。我们考虑了两种策略，它们在成本与准确率权衡上各不相同。

*3.3.1 余弦相似度检索。*最直接的方法是对输入的任务描述 d 进行嵌入，并计算其与已存储任务（及子任务）描述嵌入之间的余弦相似度。然后检索与最相似的已存储描述相关联的提示，并将其注入提示词中。这种策略在检索时无需调用大语言模型，速度快且成本低——本质上只是一次向量数据库查询。

有两种互补机制用于控制选择哪些提示：

- **相似度阈值** τ：只有来源描述与 d 的余弦相似度 $\geq \tau$ 达到要求的提示才符合条件。较高的阈值（例如，$\tau \geq 0.85$）可确保检索到的提示与当前任务密切相关，但也可能排除语义等同、措辞却不同的任务所对应的提示。例如，“我想办理 Amazon Prime 会员”和“帮我注册 Amazon Prime”描述的是同一项任务，但由于词汇差异，其余弦相似度可能低于 0.85。较低的阈值（例如，$\tau \leq 0.6$）覆盖范围更广，但也可能检索到来自无关任务的提示——“预订飞往纽约的航班”中的提示不太可能对执行“将我的日历更新到下周”的智能体有所帮助，尽管两者都涉及与日程安排相关的措辞，因此可能产生中等相似度分数。
- **首-k 选**：经过阈值筛选后，系统会选出得分最高的 k 提示。无论有多少条提示通过筛选，注入提示词的数量都会受到限制阈值，从而避免在大量已存储任务相似度适中时出现提示词膨胀。

在实际应用中，这两种机制会结合使用：系统先检索所有相似度为 $\geq \tau$ 的提示，然后按相似度分数选出排名前 k 的提示。典型取值为 $\tau \in [0.5, 0.7]$ 和 $k \in [5, 10]$。

*3.3.2 大语言模型引导的选择。*一种表达能力更强的方法是在检索时使用大语言模型分析任务描述 d，识别应用上下文和任务类别，并推断哪些类型的指导最相关。大语言模型会构建一个结构化检索查询，其中包含：

- **元数据过滤**：大语言模型会识别出，“完成我待处理的 Venmo 付款请求”这项任务涉及 Venmo 应用和支付操作，并将检索范围限定为支付领域的提示（或应用上下文为空值的通用提示）。
- **类别感知**：根据任务描述，大语言模型可能会判断恢复提示尤其相关（例如，任务提到重试一笔失败的付款），或者应优先考虑策略提示（例如，任务涉及多步骤工作流）。

大语言模型引导的选择成本更高（每项任务都需要额外调用一次大语言模型），但它能够推理纯嵌入相似度可能遗漏的细微差异。例如，大语言模型可以识别出“删除所有超过 30 天的已读电子邮件”和“通过移除旧邮件来清理收件箱”是同一项任务，即使两者的嵌入存在差异；它还可以推断，涉及“结账”的任务意味着与支付相关的提示是适用的，即使任务描述中从未提到“支付”。

比较。余弦相似度检索简单、快速，且运行时无需调用大语言模型，因此适用于对延迟敏感或成本受限的部署。大语言模型引导的选择能够更深入地推理任务上下文，但代价是需要额外调用一次大语言模型。我们在第4节中通过实证评估这两种策略。

*3.3.3 提示词集成。*无论采用何种检索策略，选中的提示都会作为“指南”部分注入智能体的提示词中，置于任务上下文之后、标准智能体指令之前。每条提示的格式都便于快速浏览和执行，并突出显示优先级、类别、可执行内容、目的、实施步骤和触发条件。例如：

```
[优先级：高] 恢复          y Tip:
登录尝试因“凭据无效”而失败时，请重新调用
supervisor.show_account_passwords()，并按目标应用名称筛选，以确认使用的是正确的应用专用密码。


适用情况：初次登录尝试后，任何应用的身份验证失败。


步骤：
1. 从监督者处重新获取凭据 2. 筛选出特定的应用名称（完全匹配） 3. 使用正确的凭据重试登录
```

这种格式有助于智能体快速识别指导的类型、优先关注关键提示，并理解要做什么以及为什么这么做。提示词集成形成了一个反馈循环：接收到相关提示的智能体能够避免失败模式、更高效地执行任务并应用成功的策略，从而生成更高质量的轨迹，进一步彰显记忆系统的价值。

4 评估

我们在 AppWorld 基准上评估了轨迹信息化记忆生成框架。AppWorld 基准是一套全面的评估套件，用于评估能够跨多个应用程序执行复杂任务的大语言模型智能体。我们的评估考察两个维度：（1）提示提取粒度的影响（任务级提示与子任务级提示）；（2）检索策略的影响（余弦相似度与大语言模型引导的选择）。评估结果表明，配备了来自过往执行过程的习得的记忆的智能体，其表现显著优于没有记忆的智能体；在具有挑战性的任务上，提升尤为明显。

4.1 实验设置

4.1.1 基准描述。 AppWorld 是一个基准，用于评估大语言模型智能体在多样化应用领域中完成现实任务的能力。该基准包含涵盖电子商务、电子邮件、日历、文件管理以及其他常见应用场景的任务。每项任务都由一条自然语言指令构成，智能体必须通过与各类应用程序提供的 API 交互来执行该指令。

该基准包含两项关键评估指标：

任务目标完成率（TGC）衡量智能体通过所有程序化单元测试的单项任务所占的百分比。这些测试用于验证 API 使用是否正确、数据库状态是否发生预期变化，以及最终状态是否符合预期。每项任务都是一项复杂的、多步骤的、基于应用的挑战，通常需要调用多个 API，平均涉及 1.8 个应用和 9.5 个 API。只有通过所有单元测试，任务才算成功。

场景目标完成度（SGC）衡量智能体正确完成某个场景所关联的所有任务变体（通常为三个）的场景所占的百分比，用于测试智能体在相关任务间表现的一致性。只有所有变体都通过，场景才计为成功，因此这项指标比 TGC 更严格。

AppWorld 中的任务按难度级别分类：

- **难度1（简单）**：要求进行基本 API 交互的简单任务，通常仅涉及单一领域，执行步骤也较为直接。
- **难度2（中等）**：复杂度适中的任务，可能涉及多个领域，或需要条件逻辑和错误处理。
- **难度3（困难）**：需要仔细规划、前置条件管理、跨领域协调和稳健错误恢复的复杂多步骤任务，通常涉及 50+ 行等效代码以及最多26个API

4.1.2 智能体配置。 我们使用单智能体配置进行评估，该配置实现了简化的 ReAct式推理与行动循环。智能体会迭代地推理当前任务状态，选择要执行的操作，通过 API调用执行这些操作，并观察结果。智能体持续这一循环，直到确定任务已完成或遇到不可恢复的故障。智能体和提示提取流水线均使用 GPT-4.1。

基础智能体（无记忆）仅接收任务指令和标准提示，其中包括其角色描述、可用API以及任务执行的一般指南。记忆增强智能体还会额外接收从记忆系统检索到的提示，这些提示会在智能体开始推理之前注入提示词。

4.1.3 提示提取配置。 我们评估两种提示提取粒度：

任务级提示 从完整轨迹中提取，如第3.1.3节所述。每条轨迹都会产生一套整体性的策略、恢复和优化提示，用于捕捉端到端执行模式。任务级提示适合捕捉总体策略（例如，“结账前验证所有前置条件”），但可能会把来自不同执行阶段的无关问题捆绑在一起。

子任务级提示 使用第3.1.4节所述的两阶段流水线进行提取。轨迹首先被划分为逻辑子任务（身份验证、数据检索、数据处理等），然后针对每个子任务独立提取提示。子任务级提示更加聚焦，并且可在共享相同子任务的任务之间复用。

两种类型的提示均由智能体在 AppWorld 训练集和开发集分区上的执行生成，并通过我们的流水线进行处理。

4.1.4 检索策略配置。 我们评估了两种检索策略，用于在运行时选择要注入智能体提示词的提示：

余弦相似度检索 使用文本嵌入模型对任务指令进行嵌入，并检索与查询嵌入余弦相似度最高的前 k 条提示。这是一种标准检索方法，在检索时无需调用大语言模型，且速度快、成本低。

大语言模型引导的选择 使用大语言模型分析任务指令，识别应用上下文和任务类别，并构建一个结合语义相似度、元数据过滤和优先级加权排序的检索查询。这种方法的表达能力更强——它能够推理哪些提示类别最相关，并确保关键提示得到呈现——但在检索时需要额外调用一次大语言模型。

对于这两种策略，都会检索前 5 条提示，并在推理开始前将其注入智能体的提示词中。

4.1.5 评估协议。我们在 AppWorld 的三个数据划分上评估了各项配置：(1) **常规测试**分区，其中包含记忆生成期间未见过的留出任务，用于衡量智能体将学习到的模式泛化到新任务的能力；(2) **训练**分区，提示由此生成，用于衡量再次遇到相同任务时提示能多有效地提升性能；(3) **开发集**分区，同样用于提示生成，可提供互补视角。

每项任务均独立执行，最多进行 30 个推理—行动步骤。任务目标和场景目标完成度通过 AppWorld 的自动评估框架进行评估，该框架

通过检查所有相关应用的最终状态，验证所有显式要求（任务目标）和隐式要求（场景目标）是否均已满足。

4.2 留出结果（常规测试）

常规测试划分包含记忆生成期间未见过的任务，为评估记忆系统将学习到的模式泛化到新任务的能力提供了最严格的测试。我们展示了多种配置下的结果。

4.2.1 使用大语言模型引导的选择的子任务级提示。 表1和表2展示了使用大语言模型引导的选择的子任务级提示结果——这是场景目标完成度表现最佳的配置。

表1：子任务提示 + 大语言模型选择：常规测试

Type	任务目标	场景目标
汇总	73.2	64.3
难度1	91.2	89.5
难度2	70.8	56.2
难度 3	58.7	47.6

表2：基线智能体（无记忆）：常规测试

Type	任务目标	场景目标
汇总	69.6	50.0
难度1	89.5	79.0
难度2	66.7	56.2
难度 3	54.0	19.1

记忆增强智能体的 TGC 达到 73.2%，而基线为 69.6%（+3.6 个百分点）；SGC 达到 64.3%，而基线为 50.0%（+14.3 个百分点）。SGC 的提升幅度更大，这表明记忆系统不仅能帮助智能体正确完成单项任务，还能显著提高同一场景下不同任务变体之间的一致性。由于 SGC 要求所有变体都通过，因此对偶发失败十分敏感，而这正是学习得到的提示有助于缓解的问题。

任务复杂度越高，收益越大。难度1任务的 TGC 提升 +1.7 个百分点，SGC 提升 +10.5 个百分点，而基线的 TGC 已经很高。难度2任务的 TGC 提升 +4.1 个百分点，SGC 则没有变化；这类任务受益于有关错误处理和前置条件验证的学习到的模式。难度 3 任务的提升最为显著：TGC 提升 +4.7 个百分点，SGC 更是提升了 +28.5 个百分点（19.1% → 47.6%），相对增幅达 149%。这些复杂任务需要精密的规划和可靠的错误恢复，而这些正是记忆系统能够提供最多指导的领域。

4.2.2 通过余弦相似度检索获取任务级提示。接下来，我们评估通过余弦相似度检索获取任务级提示的效果。任务级提示从完整轨迹中提取整体见解，而非将其分解为子任务。检索时，会将传入的任务描述转换为嵌入，并与存储的任务描述嵌入进行比较；检索相似度超过阈值 τ 的描述所对应的提示。我们评估了三种检索参数配置，以考察阈值和 top-k 选择的影响。

表 3：任务级提示 + 余弦 ($\tau \geq 0.5$，前 3 名)：常规测试

Type	任务目标	场景目标
汇总	66.7	48.2
难度1	86.0	68.4
难度2	70.8	56.2
难度 3	46.0	23.8

表 4：任务级提示 + 余弦 ($\tau \geq 0.6$)：常规测试

Type	任务目标	场景目标
汇总	72.0	62.5
难度1	91.2	84.2
难度2	72.9	68.8
难度 3	54.0	38.1

表 5：任务级提示 + 余弦 ($\tau \geq 0.5$)：常规测试

Type	任务目标	场景目标
汇总	70.2	57.1
难度1	91.2	84.2
难度2	64.6	43.8
难度 3	55.6	42.9

这三种余弦相似度配置揭示了阈值、top-k 选择与任务复杂度之间的重要交互作用。

Top-k 限制会损害性能。最严格的配置（$\tau \geq 0.5$，top-3）在汇总层面表现低于基线（TGC 为 66.7%，SGC 为 48.2%），分别下降 −2.9 pp 和 −1.8 pp。top-3 限制使智能体只能使用来自三个匹配任务描述的提示，因此可能会排除相关指导。这对复杂任务的影响尤其严重：难度 3 的 TGC 降至 46.0%（较基线下降 −8.0 pp）。

阈值 $\tau = 0.6$ 是最佳选择。采用 $\tau \geq 0.6$（不设 top-k 限制）的配置，在余弦相似度配置中取得了最出色的整体结果：TGC 为 72.0%（+2.4 pp），SGC 为 62.5%（+12.5 pp）。这一阈值实现了有效平衡：足够严格，可以排除来自无关任务的提示；又足够宽松，可以捕捉语义等同但字面表述不同的任务描述（例如，“我想要一个 Amazon Prime 会员资格”和“帮我注册 Amazon Prime”）。难度 3 的 SGC 提升十分显著：19.1% → 38.1%（+19.0 pp），相对增幅达 99%。

阈值越低，纳入的噪声越多。降至 $\tau \geq 0.5$（无 top-k）后，TGC 为 70.2%（+0.6 个百分点），SGC 为 57.1%（+7.1 个百分点）——优于基线，但两项指标均低于 $\tau \geq 0.6$。较低的阈值会纳入来自关联度较弱任务的提示，从而稀释有效信号。值得注意的是，对于难度 3，使用 $\tau \geq 0.5$ 时的 TGC（55.6%）略高于使用 $\tau \geq 0.6$ 时（54.0%），这表明对于最复杂的任务，扩大检索范围偶尔能从关联较弱的任务中找到有用的提示。然而，难度 2 呈现相反趋势（64.6% 对 72.9%）；此时，无关提示带来的噪声影响更大。

4.2.3 使用余弦相似度检索的子任务级提示。为了将检索策略的影响与提示粒度的影响区分开来，我们还评估了使用余弦相似度检索的子任务级提示（$\tau \geq 0.6$，无 top-k）——检索参数与效果最佳的任务级余弦配置相同，但改用子任务级提示。

表6：子任务提示 + 余弦（$\tau \geq 0.6$）：常规测试

Type	任务目标	场景目标
汇总	73.8	57.1
难度1	91.2	73.7
难度2	72.9	56.2
难度 3	58.7	42.9

这一配置的 TGC 达到 73.8%（比基线高 +4.2 个百分点），是所有配置中最高的 TGC；SGC 则为 57.1%（+7.1 个百分点）。将其与采用大语言模型引导的选择的子任务级提示（表1）进行比较，可以在保持提示粒度不变的情况下，单独考察检索策略的影响：使用余弦检索时，TGC 略高（73.8% 对 73.2%），但 SGC 大幅下降（57.1% 对 64.3%，相差 7.2 个百分点）。这种差异在难度 3 上最为显著，SGC 从 47.6% 降至 42.9%。大语言模型引导的选择能够对任务上下文进行推理并确定提示类别的优先级，这一能力对于跨变体一致性似乎至关重要；尽管简单的余弦检索已足以完成单个任务（且略有优势）。

4.2.4 配置比较。表7比较了留出的常规测试分区上的所有配置；对于两种余弦相似度配置，均使用 $\tau \geq 0.6$ （无 top-k）。

这三种配置清晰地揭示了影响任务目标完成率和场景目标完成度的因素各不相同。

提示粒度影响 TGC。 无论采用哪种检索策略，子任务级提示在 TGC 上都优于任务级提示：余弦检索下为 73.8%，LLM 引导下为 73.2%，而任务级余弦检索为 72.0%。将内容进一步细分为可复用的子任务模式，能为完成单项任务提供更有针对性的指导；这一点在难度 3 的任务中尤为明显，子任务级提示的 TGC 为 58.7%，而任务级提示为 54.0%（+4.7 个百分点）。

检索策略影响 SGC。 与相同提示粒度下的余弦相似度检索相比，LLM 引导的选择显著提升了场景目标完成度：子任务级提示分别为 64.3% 和 57.1%（+7.2 个百分点）。这种差距在不同难度等级中均一致，其中难度1的差距最大（89.5% 对 73.7%，+15.8 个百分点）。大语言模型能够推理任务上下文、确定提示类别的优先级并应用元数据过滤，从而在同一场景内的不同任务变体中提供更一致的 指导，减少 SGC 会予以惩罚的偶发失败。

交互效应。 有趣的是，尽管任务级提示的 TGC 较低，但采用余弦相似度的任务级提示取得了更高的 SGC（62.5%），高于采用余弦相似度的子任务级提示（57.1%）。任务级提示编码了整体性的端到端策略，能够促进相关任务变体采用一致的执行模式；而子任务级提示虽然对完成单个任务更为精准，却可能为同一场景的不同变体检索到不同的子任务提示子集，从而引入行为差异。LLM引导的选择则通过推理整体任务上下文并确保不同变体间的提示选择一致，弥补了这一不足。

所有配置的表现都显著优于基线，证实无论选择何种具体配置，记忆系统都能带来切实价值。最佳配置取决于部署目标：若追求最佳整体表现，可采用子任务级提示和大语言模型引导的选择；若希望以较低的检索成本获得最高的单任务准确率，可采用子任务级提示和余弦相似度；若希望在不产生大语言模型检索开销的情况下实现良好平衡，则可采用任务级提示和余弦相似度。

4.3 源分区结果（训练集和开发集）

训练集和开发集分区用于提示生成：从智能体在这些任务上的轨迹中提取提示。这些分区上的结果衡量的是一种不同于常规测试的场景：当智能体再次遇到相同或结构上相同的任务时，如果利用其先前执行中提取的提示进行增强，会发生什么？该设置评估记忆系统在重复任务上实现自我改进的能力，与在常规测试集上进行的泛化评估相辅相成。

表 8–11 展示了源数据划分上采用子任务级提示和大语言模型引导的选择时的结果。

正如预期，在源数据划分上的提升大于常规测试：训练集上 TGC 提升 +4.4 个百分点 / SGC 提升 +10.0 个百分点，开发集上 TGC 提升 +12.3 个百分点 / SGC 提升 +26.3 个百分点。当智能体遇到与提示来源任务在结构上相似的任务时，这些提示最能切合上下文，因此出现更大的提升也在预期之中。

有两种与数据划分相关的模式值得注意。在训练集的难度 1 任务中，基线已达到 100%，而增强记忆的智能体得分略低（TGC 为 94.4%，SGC 为 83.3%），这表明，对于智能体已表现最佳的简单任务，注入额外提示可能会造成轻微干扰。在开发集上，难度 3 的基线在 TGC 和 SGC 上均已达到 100%，因此开发集的汇总提升（TGC 提升 +12.3 个百分点，SGC 提升 +26.3 个百分点）完全来自难度 1 和 2 的提升。在这两种情况下，从记忆中受益的任务所带来的整体提升，都远远超过了性能上限或干扰效应的影响。

4.4 跨配置总结

表 12 汇总了在全部三个数据划分中，采用大语言模型引导的选择所带来的子任务级提示总体改进。

可以观察到若干现象。首先，记忆系统在全部三个数据划分上均提升了性能，这证实收益并不局限于生成这些提示的任务。常规测试增益（+3.6 TGC， +14.3 SGC）表明对未见任务具有真正的泛化能力。其次，源数据划分呈现出更大的 TGC 提升，这符合预期——当智能体再次遇到派生出这些提示的任务时，提示在上下文中最为相关。有趣的是，常规测试的 SGC 增益（+14.3 个百分点）超过了训练的 SGC 增益（+10.0 个百分点），这表明子任务级分解与 LLM 引导的检索在提升跨变体一致性方面泛化得特别好。第三，在所有数据划分上，SGC 的提升始终超过 TGC 的提升，表明记忆系统在提升任务变体间一致性方面尤为有效。恢复

表7：常规测试（汇总）上的配置比较

提示粒度	检索策略	TGC	Δ TGC	SGC	Δ SGC
基线（无记忆）		69.6	—	50.0	—
子任务级	大语言模型引导的选择	73.2	+3.6	**64.3**	**+14.3**
子任务级	余弦相似度 ($\tau \geq 0.6$)	**73.8**	**+4.2**	57.1	+7.1
任务级	余弦相似度 ($\tau \geq 0.6$)	72.0	+2.4	62.5	+12.5

表8：子任务提示 + 大语言模型选择：训练

Type	任务目标	场景目标
汇总	91.1	83.3
难度1	94.4	83.3
难度2	88.9	83.3
难度 3	88.9	83.3

表 9：基线智能体（无记忆）：训练

Type	任务目标	场景目标
汇总	86.7	73.3
难度1	100.0	100.0
难度2	77.8	58.3
难度 3	77.8	50.0

表10：子任务提示 + 大语言模型选择：开发集

Type	任务目标	场景目标
汇总	89.5	73.7
难度1	90.0	80.0
难度2	87.5	62.5
难度 3	100.0	100.0

表11：基线智能体（无记忆）：开发集

Type	任务目标	场景目标
汇总	77.2	47.4
难度1	80.0	60.0
难度2	70.8	25.0
难度 3	100.0	100.0

提示与策略提示编码了前置条件验证和错误处理模式，从而降低行为差异，使智能体能够可靠地完成所有变体，而不是在某些变体上成功、在另一些上失败。

5 相关工作

我们的工作处于智能体记忆系统、基于轨迹的学习以及自我改进智能体的交汇处。我们从三个方面梳理相关工作：大语言模型智能体的记忆架构、从执行轨迹中学习的系统，以及通过经验实现智能体自我改进的方法。

5.1 记忆分类体系与架构

近期有两篇综述全面梳理了基于大语言模型的智能体记忆分类体系。Zhang等人 [17] 从三个维度组织设计空间——记忆来源（智能体与环境的交互、内部推理、用户反馈）、记忆形式（自然语言、嵌入、数据库、结构化知识）和记忆操作（读取、写入、反思、管理）——并指出现有工作的主要局限：表征过于简单，决定记住或遗忘什么的操作不够成熟，以及评估工作零散。Du等人 [4] 则从互补的、以操作为中心的视角出发，定义了六种原子记忆操作：巩固、更新、索引、遗忘、检索和压缩。按照他们的术语，我们的提示提取属于一种巩固（将原始轨迹转化为抽象提示），提示精炼属于更新，而选择性保留属于遗忘。两篇综述都指出，现有大多数系统存储的是原始或仅经简单处理的文本，缺乏有效的智能体记忆所需的结构化抽象和质量意识型整理。我们的框架直接针对这些已指出的不足。

5.2 语义记忆系统

应用最广泛的智能体记忆系统运行于语义层面，存储从交互中提取的事实知识。Mem0 [2] 会从对话中提取并整合事实片段——用户偏好、实体和关系——并存入向量存储，从而在对话个性化方面实现出色的延迟表现和令牌效率。A-MEM [15] 引入了一种受卡片盒笔记法启发的自组织记忆架构，其中每条记忆都以结构化笔记的形式存储，附带上下文描述、关键词以及指向相关记忆的显式链接，由此形成一个涌现的知识网络。尽管这两个系统都针对各自用途进行了精心设计，但它们从根本上存储的是陈述性知识（已知的内容），而非程序性 或经验性 知识（该做什么，以及从实践中学到了什么）。它们没有分析执行轨迹、对失败进行因果归因或生成分类行为指导的机制。我们的框架通过从执行经验中提取结构化、可执行的提示，而非对话事实，弥补了这一缺口。

5.3 从执行轨迹中学习

越来越多的研究开始探讨智能体如何从过去的执行轨迹中学习，这与我们的贡献最为直接相关。

工作流与流程提取。Agent 工作流记忆（AWM） [13] 从网页导航中的成功智能体轨迹中提取可复用的多步骤工作流，提升幅度分别达到 24.6% 和

表 12：总体改进汇总：子任务提示 + 大语言模型选择

分区	任务目标（基线）	任务目标（+记忆）	场景目标 场景目标（基线）	（+记忆）
常规测试	69.6	73.2（+3.6）	50.0	64.3 (+14.3)
训练	86.7	91.1 (+4.4)	73.3	83.3 (+10.0)
Dev	77.2	89.5 (+12.3)	47.4	73.7 (+26.3)

51.1%（相对提升），分别适用于 Mind2Web 和 WebArena。AWM 展示了显著的“滚雪球效应”：简单工作流可以组合成更复杂的工作流。然而，AWM 仅从成功的轨迹中学习——它没有从失败、恢复或低效执行中吸取经验的机制。Mem^p [5] 将程序性记忆视为首要优化对象，系统地探索如何从轨迹中构建记忆、检索相关程序，以及随时间更新条目。虽然 Mem^p 涵盖了完整的记忆生命周期，但它关注的是程序性指令（“如何做 X”），而不是我们的提示类别所捕捉的诊断性行为洞见（“出了什么问题以及原因”）。AgentRR [6] 借鉴了软件工程中的记录与回放范式，记录完整的智能体交互轨迹，并将其总结为结构化经验，以供未来回放。与 AWM 一样，它主要从成功执行中学习。

推理与策略提取。 ReasoningBank [9] 是与我们最为接近的研究之一，它从智能体自行判定成功与失败的经历中提炼出可泛化的推理策略。它与我们的共同见解是，智能体应从成功和失败中学习。关键区别在于抽象层次：ReasoningBank侧重于元认知推理策略，而我们的提示则侧重于从特定执行模式中提炼具体的行为指导。这两种方法相辅相成。

上下文工程与自我改进。 ACE（智能体上下文工程） [16] 将智能体的上下文视为不断演进的“策略手册”，通过生成—反思—整理循环积累并改进策略，并在AppWorld上实现了10.6个百分点的提升。我们的框架与ACE在若干方面有所不同：我们生成具有类型化类别（策略、恢复、优化）和丰富元数据的结构化记忆条目，并进行选择性检索，而非将不断演进的文本文件全文纳入上下文；我们通过明确的因果归因，将结果追溯至具体决策；并维护从提示到源轨迹的溯源追踪。

基于学习式检索的经验回放。Memento [18] 提出了一种记忆增强型马尔可夫决策过程形式化方法，其中由学习得到的神经策略为给定任务选择要检索的已存储轨迹。然而，Memento 存储的是原始轨迹，并未将其提炼为可迁移的见解——将轨迹巩固为可执行经验的工作被留给了大语言模型的上下文内推理。

5.4 实证基础

Xiong 等人[19]为基于轨迹的记忆系统提供了关键的实证依据，指出了经验跟随属性以及两种失效模式：错误传播和错位的经验重放。他们发现，将选择性添加与选择性删除相结合，较朴素的记忆增长方式带来了 10% 的绝对性能提升。这些发现直接促使我们采用结构化方法：通过提取带有明确适用条件的抽象提示，而非存储原始轨迹，并利用元数据对提示进行分类，以实现精准的上下文匹配，我们的框架缓解了这两种失效模式。

6 结论

我们提出了一种框架，可从大语言模型智能体执行轨迹中自动提取可执行的经验，并将其存储为结构化记忆提示，以提升智能体未来的表现。我们的四组件流水线——轨迹智能提取、决策归因分析、上下文学习生成和自适应记忆检索——涵盖了失败、恢复、低效成功和顺利成功中的各类学习机会。在 AppWorld 基准上的评估表明，该框架能够持续带来提升：在留出任务上的场景目标完成率最高提升 14.3 个百分点；在复杂多步骤任务上的效果尤为显著（SGC 提升 28.5 个百分点，相对增幅为 149%）。该框架可自然扩展到多智能体系统，结合跨智能体归因和面向智能体角色的指导；这部分工作留待未来开展。我们还计划使用更多先进的开源模型（例如通义千问 [12] 和 GPT-OSS [1]）评估该框架，以考察提示质量和检索效果在不同模型系列中的差异。本文介绍的技术正在应用于 IBM 的可配置通用智能体（CUGA） [7, 8] 平台，用于构建和部署企业级智能体系统；借助基于轨迹的记忆，智能体能够从运营经验中持续改进。

参考文献

[1] 待办。2025。添加 GPT-OSS 参考文献。占位内容——请替换为正确的 GPT-OSS 引文。[2] Prateek Chhikara、Dev Khant、Saket Aryan、Taranjeet Singh 和 Deshraj Yadav。2025。Mem0：借助可扩展长期记忆构建可用于生产环境的 AI 智能体。*arXiv* 预印本 *arXiv:2504.19413*（2025）。[3] Chad DeChant。2025。AI 智能体中的情景记忆带来了应当研究并缓解的风险。*arXiv* 预印本 *arXiv:2501.11739*（2025）。[4] Yiming Du、Wenyu Huang、Danna Zheng、Zhaowei Wang、Sébastien Montella、Mirella Lapata、Kam-Fai Wong 和 Jeff Z. Pan。2025。重新思考 AI 中的记忆：分类、运作、主题与未来方向。*arXiv* 预印本 *arXiv:2505.00675*（2025）。[5] Runnan Fang、Yuan Liang、Xiaobin Wang、Jialong Wu、Shuofei Qiao、Pengjun Xie、Fei Huang、Huajun Chen 和 Ningyu Zhang。2025。Mem^p：探索智能体程序性记忆。*arXiv* 预印本 *arXiv:2508.06433*（2025）。[6] Erhu Feng、Wenbo Zhou、Zibin Liu、Le Chen、Yunpeng Dong、Cheng Zhang、Yisheng Zhao、Dong Du、Zhichao Hua、Yubin Xia 和 Haibo Chen。2025。从实践中获取经验：具备记录与回放功能的大语言模型智能体。*arXiv* 预印本 *arXiv:2505.17716*（2025）。[7] IBM。2025。CUGA：可配置通用智能体。https://github.com/cuga-project/cuga-agent。[8] SamiMarreed、AlonOved、Avi Yaeli、Segev Shlomov、IdoLevy、OfferAkrabi、Aviad Sela、Asaf Adi 和 Nir Mashkif。2025。迈向企业级计算机使用

通用智能体。*arXiv* 预印本 *arXiv:2503.01861*（2025）。[9] Siru Ouyang、Jun Yan、I-Hung Hsu、Yanfei Chen、Ke Jiang、Zifeng Wang、Rujun Han、Long T. Le、Samira Daruki、Xiangru Tang、Vishy Tirumalashetty、George Lee、Mahsan Rofouei、Hangfei Lin、Jiawei Han、Chen-Yu Lee 和 Tomas Pfister。2025。ReasoningBank：通过推理记忆扩展智能体自我进化。*arXiv* 预印本 *arXiv:2509.25140*（2025）。[10] Charles Packer、Sarah Wooders、Kevin Lin、Vivian Fang、Shishir G. Patil、Ion Stoica 和 Joseph E. Gonzalez。2023。MemGPT：迈向作为操作系统的大语言模型。*arXiv* 预印本 *arXiv:2310.08560*（2023）。[11] Mathis Pink、Qinyuan Wu、Vy Ai Vo、Javier Turek、Jianing Mu、Alexander Huth 和 Mariya Toneva。2025。观点：情景记忆是长期大语言模型智能体缺失的关键环节。*arXiv* 预印本 *arXiv:2502.06975*（2025）。[12] Qwen 团队。2025。Qwen2.5 技术报告。*arXiv* 预印本 *arXiv:2412.15115*（2025）。[13] Zora Zhiruo Wang、Jiayuan Mao、Daniel Fried 和 Graham Neubig。2024。智能体工作流记忆。*arXiv* 预印本 *arXiv:2409.07429*（2024）。[14] Zidi Xiong、Yuping Lin、Wenya Xie、Pengfei He、Jiliang Tang、Himabindu Lakkaraju 和 Zhen Xiang。2025。记忆管理如何影响大语言模型

智能体：经验遵循行为的实证研究。*arXiv* 预印本 *arXiv:2505.16067*（2025）。[15] Wujiang Xu、Zujie Liang、Kai Mei、Hang Gao、Juntao Tan 和 Yongfeng Zhang。2025。A-MEM：面向大语言模型智能体的智能体记忆。*arXiv* 预印本 *arXiv:2502.12110*（2025）。[16] Qizheng Zhang、Changran Hu、Shubhangi Upasani、Boyuan Ma、Fenglu Hong、Vamsidhar Kamanuru、JayRainton、Chen Wu、MengmengJi、Hanchen Li、Urmish Thakker、James Zou 和 Kunle Olukotun。2025。智能体上下文工程：为自我改进语言模型演进上下文。*arXiv* 预印本 *arXiv:2510.04618*（2025）。[17] Zeyu Zhang 、 Xiaohe Bo 、 Chen Ma 、 Rui Li 、 Xu Chen、Quanyu Dai、Jieming Zhu、Zhenhua Dong 和 Ji-Rong Wen。2025。基于大语言模型的智能体记忆机制综述。*ACM 信息系统汇刊*（*TOIS*）（2025）。doi:10.1145/3748302 arXiv:2404.13501。[18] Huichi Zhou、Yihang Chen、Siyuan Guo、Xue Yan、Kin Hei Lee、Zihan Wang、Ka Yiu Lee、Guchun Zhang、Kun Shao、Linyi Yang 和 Jun Wang。2025。Memento：无需微调大语言模型即可微调大语言模型智能体。*arXiv* 预印本 *arXiv:2508.16153*（2025）。