

EnvAct: 基于大语言模型与结构化环境表征的 物联网自动化执行框架

胡明明¹, 滕储华¹, 李明辉¹, 孙丙坤¹, 沈立炜^{1,2*}

(1. 复旦大学计算与智能创新学院, 上海 200433; 2. 山东省复杂网络软件自动构造技术重点实验室, 山东济南 250101)

摘要: 随着物联网设备在智能家居、智慧建筑等场景中广泛部署, 智能系统需要在设备状态持续变化、用户需求表达模糊和外部环境动态扰动的环境下完成可靠控制。现有触发-动作规则方法多依赖预先设定的静态逻辑, 难以处理设备操作的前置约束、状态依赖和运行时环境变化等问题; 而基于大语言模型(Large Language Model, LLM)的物联网代理虽然增强了自然语言理解和任务规划能力, 但通常缺乏统一的结构化环境表征, 容易产生环境依赖遗漏、设备选择错误或操作顺序不合理等问题。尤其在多设备协同、持续监测和异常反馈场景中, 上述问题会进一步影响任务执行的稳定性。为解决上述问题, 本文提出 EnvAct (Environment-aware Action framework), 一种基于 LLM 与结构化环境表征的物联网自动化执行框架。EnvAct 将空间、设备、状态属性、操作约束和状态依赖统一建模, 构建由约束表和依赖表组成的环境表征机制。约束表描述设备操作执行前需要满足的前置状态、操作效果及状态生效延迟; 依赖表建立环境状态到候选操作的逆向映射, 使系统能够在前置条件不满足时自动补全必要操作。为降低冷启动建模成本, 本文从 Home Assistant 社区蓝图脚本中抽取事件-条件-动作规则, 并利用 LLM 结合当前设备、可观测状态和物理常识过滤个性化或不适配规则, 生成可持续更新的环境表征。系统由规划器、合规器、执行器、反思器和环境表征层组成。规划器融合用户需求、实时环境、设备接口和历史偏好生成初始操作序列; 合规器基于约束表和依赖表进行合法性校验、依赖补全和偏序图构建; 执行器按照偏序关系并行调度独立操作, 并通过重试、反馈和状态验证实现闭环执行; 反思器根据执行轨迹和用户反馈修正环境表征与偏好信息。实验结果表明, 在六类物联网任务中, EnvAct 在 qwen-plus 和 GPT-4o 驱动下的总体成功率分别达到 88.3% 和 90.0%, 优于 Sasha 和 SAGE 等基准方法; 在动态干扰实验中, 系统平均执行可靠性达到 81.25%; 在真实沙盘实验中, 系统成功率保持在 90% 以上, 平均操作响应时间低于 30 s。结果表明, EnvAct 能够提升动态物联网场景下自动化任务执行的可靠性、适应性和工程可用性。

关键词: 物联网; 大语言模型; 自动化执行; 环境表征; 闭环控制; 智能系统

基金项目: 山东省重点研发计划(No.2025CXPT073)

中图分类号: TP273; TP18

文献标识码: A

文章编号: 0372-2112(2026)06-2657-19

电子学报 URL: <http://www.ejournal.org.cn>

DOI: 10.12263/DZXB.20250886

EnvAct: An IoT Automation Execution Framework Based on Large Language Models and Structured Environment Representation

HU Mingming¹, TENG Chuhua¹, LI Minghui¹, SUN Bingkun¹, SHEN Liwei^{1,2*}

(1. College of Computer Science and Artificial Intelligence, Fudan University, Shanghai 200433, China;

2. Shandong Key Laboratory of Complex Network Software Automatic Construction Technology, Jinan, Shandong 250101, China)

Abstract: With the widespread deployment of internet of things (IoT) devices in smart homes, smart buildings, and other application scenarios, intelligent systems are increasingly expected to provide reliable automation control in environments where device states keep changing, user requirements are often ambiguous, and external disturbances may occur during execution. Existing trigger-action rule methods mainly rely on predefined static logic. Therefore, they have limited ability to represent the preconditions of device operations, state dependencies among devices and environmental attributes, and runtime environmental changes. Large language model (LLM)-based IoT agents improve natural language understanding and task planning, but they usually lack a unified structured environmental representation. As a result, such agents may miss important environmental dependencies, select inappropriate devices, or generate operation sequences with unreasonable execution orders. These limitations become more serious in scenarios involving multi-device collaboration, continuous monitoring, and abnormal execution feedback, where they may directly affect the stability of task execution. To address these problems, this paper proposes environment-aware action framework (EnvAct), an IoT automation execution framework based on

LLMs and structured environmental representation. EnvAct uniformly models spaces, devices, state attributes, operational constraints, and state dependencies, and constructs an environmental representation mechanism composed of a constraint table and a dependency table. The constraint table describes the preconditions that must be satisfied before a device operation is executed, the expected effects of the operation, and the latency for the corresponding state effects to take place. The dependency table establishes an inverse mapping from environmental states to candidate operations, so that the system can automatically supplement necessary operations when some preconditions are not satisfied. To reduce the modeling cost in the cold-start stage, this paper extracts event-condition-action rules from Home Assistant community blueprint scripts. Based on the current devices, observable states, and physical commonsense, LLMs are used to filter personalized or environment-incompatible rules and generate an environmental representation that can be continuously updated. The EnvAct system consists of a planner, a compliance checker, an executor, a reflector, and an environmental representation layer. The planner integrates user requirements, real-time environmental information, device interfaces, and historical preferences to generate an initial operation sequence. The compliance checker performs legality verification, dependency completion, and partial-order graph construction according to the constraint table and the dependency table, thereby transforming the initial sequence into an executable plan that satisfies environmental constraints. The executor schedules independent operations in parallel according to partial-order relations, and realizes closed-loop execution through retries, execution feedback, and state verification. The reflector revises environmental representation and preference information based on execution traces and user feedback, enabling the system to improve its behavior in subsequent tasks. Experimental results show that, across six types of IoT tasks, EnvAct achieves overall success rates of 88.3% and 90.0% when driven by qwen-plus and GPT-4o, respectively, outperforming baseline methods such as Sasha and SAGE. In dynamic disturbance experiments, the average execution reliability of the system reaches 81.25%. In real sandbox experiments, the system maintains a success rate above 90%, with an average operation response time below 30 s. These results demonstrate that EnvAct can improve the reliability, adaptability, and engineering usability of automation task execution in dynamic IoT scenarios.

Keywords: internet of things; large language model; automated execution; environmental representation; closed-loop control; intelligent system

Foundation Item(s): Shandong Provincial Key Research and Development Program (No.2025CXPT073)

0 引言

近年来,物联网(Internet of Things, IoT)技术的迅猛发展推动了大量智能设备与传感器在日常生活与工业环境中的广泛部署,构成了智能家居、智慧城市、智能制造等应用场景的关键基础^[1]。为实现设备的自动控制,基于规则的系统(如IFTTT、Home Assistant)被广泛采用,用户通常通过预设的“触发-动作”逻辑(Trigger-Action)来管理设备行为^[2]。然而研究表明,当设备种类增多、规则复杂化或环境动态性增强时,其表达能力和可扩展性受到明显限制^[3];同时用户仍需理解设备状态、触发条件与动作后果,才能有效定义规则,否则容易产生误触发或行为偏差^[4]。

得益于大语言模型(Large Language Models, LLMs)在自然语言理解与推理方面的突破^[5], LLM驱动的智能系统在物联网场景中展现出广泛的应用潜力。相比传统规则系统,这类系统具备更强的泛化能力与开放性,能够以自然语言作为统一接口,灵活适配异构设备控制与复杂任务表达,为构建更智能、更自然的人机交互方式提供了新路径^[6]。尽管智能系统在文本理解与任务规划方面表现出显著优势,但其在动态环境中仍面临关键挑战:在IoT场景中,设备

与环境状态不断变化,若LLM仅基于静态上下文进行推理,往往会导致规划偏差,甚至造成任务失败^[7-8]。例如,在天气晴朗的白天,若因窗帘关闭而导致室内光照不足,相较于开灯,打开窗帘更为合理;而在深夜场景,面对相同的照明需求,开启灯光则是更为恰当的选择。

已有研究(如SAGE^[9])尝试通过引入外部工具链增强LLM的感知与执行能力。这类方法通常采用“按需调用+步进反馈”的形式,将环境状态嵌入交互流程。尽管这类方法具备一定灵活性,但其环境建模通常较为碎片化,缺乏对设备间关联关系的刻画,同时反馈链路较长,难以满足IoT环境对实时性、一致性与执行稳定性的高要求^[10-12]。另一方面,现有系统往往忽视设备执行所需的前置约束。以Sasha^[13]为例,其通过生成触发-动作规则(Trigger-Action Programming, TAP)实现自动化,但其生成的规则逻辑在执行前即已固化,本质上是一种“离线规划-盲目执行”的开环模式,缺乏针对环境动态变化的感知与自适应调节能力。一旦环境状态发生规划外的偏移,系统通常只能依赖滞后的人工干预进行事后修正,难以应对物联网场景的动态环境变化。

为应对上述挑战,本文提出一种基于大语言模型

与结构化环境表征的物联网自动化执行框架——EnvAct (Environment-aware Action framework)。相较于现有工作, EnvAct 引入了一种“规划-校正-闭环执行”的深度对齐逻辑, 重点构建了执行过程中的自动化合规校正机制。该机制通过约束表 (Constraint Table) 与依赖表 (Dependency Table) 对设备间的物理逻辑与状态关系进行解耦建模。执行器不再盲目投递初始指令, 而是通过在执行前瞬间, 由合规器根据实时环境快照, 对操作序列进行物理语义补全与合法性验证。这种基于运行时对齐的校正逻辑, 确保了语义层的抽象指令能够无缝适配物理层的约束要求, 显著提升了系统在动态场景下的可靠性。

EnvAct 框架由规划器、执行器、合规器、反思器和环境表征等模块构成, 具备端到端的任务理解与环境适应能力。我们在多个真实 IoT 场景中对该框架进行了系统性评估, 结果显示, 相较于现有基于工具链或静态规则生成的方法, 本框架在任务成功率、响应时间、可靠性等关键性能指标上均表现出显著优势。

本文的主要贡献如下: (1) 提出了环境表征机制, 通过依赖表和约束表对环境状态及设备间的依赖关系进行结构化建模, 为任务执行提供了可靠的支持; (2) 基于结构化环境表征, 提出了 EnvAct 框架, 深度融合了实时环境表征与大语言模型的推理能力, 旨在实现任务方案的合规处理, 并确保语义推理与物理环境的自适应对齐; (3) 设计并实现了基于该框架的原型系统, 在数据集上进行了实验, 并在真实沙盘环境中开展了系统性评测。

1 相关工作

1.1 物联网应用自动化

物联网自动化研究早期主要采用规则驱动方法^[14], 其核心思想是通过预先定义触发条件与动作执行逻辑, 实现设备的自动控制^[2]。典型代表包括 IFTTT^[15]、Home Assistant^[16]等平台。这些系统为非专业用户提供了直观的配置方式, 具备良好的可解释性和透明度^[17]。然而, 由于触发条件与动作逻辑均需人工设定, 用户需要投入大量时间进行规则编写和调试^[18]; 与此同时, TAP 规则通常以静态形式存储, 难以表达复杂的时空依赖或上下文约束, 导致在多设备协同和动态环境下应用时存在明显局限^[19]。

为了降低用户的配置负担, 语音助手驱动的自动化逐渐兴起^[20]。商业化产品如 Google Assistant^[21]、Alexa^[22]、Siri^[23]借助自然语言处理技术, 为用户提供了更自然的交互方式。这些系统能够通过意图识别直接控制智能家居设备, 从而显著提升了系统的可用性和普及度^[24]。在简单任务 (如单设备开关控制)

中, 这类助手表现良好。但在需要推理隐含意图、涉及跨设备协作或跨时间推理的场景下, 语音助手往往因依赖静态意图解析而失效, 表现出交互灵活性不足、上下文建模有限等问题^[25]。

在规则与语音助手之外, 研究者也探索了数据驱动的自动化方法。这类方法通常通过学习用户行为模式^[26]或直接将自然语言映射为控制逻辑^[27]来减轻用户的手动配置负担。例如, 已有工作通过挖掘历史用户行为和社区规则来推荐自动化方案^[28], 或利用序列到序列模型将自然语言指令转换为 TAP 规则^[29]。这些方法在一定程度上提升了自动化水平, 使系统能够处理部分复杂场景。然而, 其对训练数据的过度依赖导致在面对新设备类型或动态环境时泛化能力有限^[30]; 同时, 在实时性与稳定性方面也存在显著挑战。例如, FedRule^[31]揭示了在异构环境中基于数据的推荐方法存在固有困难, 而相关研究^[32]进一步表明设备属性差异会显著降低此类方法的适用性。

近年来, 随着大语言模型技术的发展, 研究者开始探索将其作为智能家居系统的核心, 以解决上述不足^[33]。Sasha^[13]展示了 LLM 在理解模糊指令、推理复杂需求方面的潜力, 其流水线结构使得模型能够根据用户目标生成合理的执行策略。然而, Sasha 的推理过程较为固定, 缺乏灵活性。SAGE^[9]在此基础上进一步引入了 LLM 驱动的动态执行规划, 使得智能体能够在任务执行中根据上下文调整操作顺序, 显著提升了系统的开放性与适应性。尽管如此, 这类方法往往依赖于碎片化的环境感知, 缺乏统一的、结构化的环境建模机制。当设备状态、空间关系或用户意图频繁变化时, 现有 LLM 代理难以保证其推理与执行过程始终与实时语境保持一致, 从而可能导致规划偏差或任务失败^[34]。

1.2 基于大语言模型的自动化执行框架

在更广泛的人工智能研究中, 基于 LLM 的自动化执行系统已成为一个快速发展的方向, 旨在通过自然语言接口规划并执行复杂任务^[35]。大量研究尝试提升其推理与规划能力, 其中, 链式思维 (Chain-of-Thought, CoT) 方法借助显式的推理链条, 将复杂任务分解为可解释的中间步骤, 显著增强了模型的推理透明性与可解性^[36]。CoT 的思想也被应用于任务规划与决策中, 使得 LLM 能够以逐步推理的方式处理多约束问题^[37]。然而, 这类方法通常假设环境是静态, 虽然在较短的视野问题上可以更成功, 但 LLM 产生的解决方案往往远非最佳^[38]。

为突破这一限制, 研究者提出了自一致性 CoT^[39]。该方法通过多路径采样生成多个推理过程,

则依赖于设备操作的执行。

传统方法通常聚焦于设备状态属性的记录,而未能充分考虑设备间复杂的先后次序与互斥关系。本研究通过引入设备与环境状态之间的约束与依赖机制,不仅实现设备操作的有序下发和执行,还能有效管理设备间的动态交互,提高系统的鲁棒性和适应性。同时,现有的主流方法往往使用状态机构建设备操作之间的关系,将环境表征转换成包含全部环境状态和状态转移的状态机,从而随着空间状态和设备数量的增多而导致状态空间爆炸问题,计算复杂度过高而无法求解。我们通过将设备操作间的协同关系解耦为设备操作与环境状态的约束关系,不仅为设备操作的并行推进提供有效支持,还显著降低了状态空间爆炸的风险,确保在复杂环境中高效求解和执行。

2.1.1 环境表征元模型与实例化

环境表征的核心元模型由空间、设备、属性、约束表和依赖表等基本元素构成,用于对智能环境中的物理实体、状态信息及其相互关系进行结构化描述。各类元素之间通过 hasDevice、hasProperty、deployedIn、hasConstraint 和 dependsOn 等关系进行连接,从而形成面向设备操作与环境状态交互的统一表示框架。

其中,空间用于刻画环境的拓扑结构,描述设备及相关属性所处的位置关系。空间节点可以通过 belongTo 和 connectWith 等关系与其他空间节点建立层级或连通关系,从而形成环境的空间布局。设备表示实际部署在物理环境中的硬件实体,其与所在空间之间通过 deployedIn 关系连接,并可通过 hasProperty 关系关联多个属性,以表征设备的功能状态和运行特征。属性用于描述设备或空间的静态与动态特性,如温度、湿度、开关状态等,是环境状态建模的基本单元。

在此基础上,约束表用于描述设备操作执行前所需满足的状态条件或限制,并通过 hasConstraint 关系与相应设备或操作关联;依赖表则用于刻画环境状态受设备操作影响的关系,并通过 dependsOn 关系对属性与操作之间的依赖关系进行建模。通过约束表和依赖表,元模型不仅能够表示环境中对象之间的静态关联,还能够描述设备操作与环境状态变化之间的动态影响关系。

基于上述元模型,环境表征可将空间、设备和属性等抽象元素实例化为具体场景对象,并通过 hasProperty、deployedIn、hasConstraint 和 dependsOn 等关系形成结构化表示。以智能建筑为例,客厅、厨房等可表示为空间节点,空调、智能灯等可表示为设备节点,温度、湿度和窗户开闭状态等可表示为属性节点。对于“空调打开操作”,其前置约束可设为“窗户

处于关闭状态”;而该状态可由“窗户关闭操作”产生,因此二者分别记录于“空调打开操作”的约束表和“窗户关闭状态”的依赖表中,为后续需求解析、依赖补全和设备协同执行提供依据。

2.1.2 约束表与依赖表的初始化构建

约束表与依赖表是 EnvAct 框架实现逻辑对齐的核心数据结构,二者构成了一对互为对偶的映射,描述了设备操作与环境状态之间的因果链条。

(1) 形式化定义

约束表 (Constraint Table) 刻画了操作对环境状态的依赖,对于任一操作 $op \in A$, 其中 A 表示所有可执行的设备操作集合,约束表定义如式(1)所示:

$$C(op) = (\text{Pre}(op), \text{Eff}(op), \text{Latency}(op)) \quad (1)$$

其中: $\text{Pre}(op)$ 表示操作的前置状态集合,既包括正条件(如 $\text{Window} = \text{Closed}$),也包括负条件(如 $\text{AC}! = \text{On}$),互斥关系被统一建模为负前置条件; $\text{Eff}(op)$ 表示操作的效果,即对环境状态的变更; $\text{Latency}(op)$ 则描述操作效果产生的延迟时间。

依赖表 (Dependency Table) 刻画了状态到操作的逆向映射。设 A 为所有可执行的设备操作集合, Σ 表示环境表征层中所有状态谓词的集合,例如 $\text{Window} = \text{Closed}$, $\text{AC} = \text{On}$, $\text{Temperature} < 6^\circ\text{C}$ 等。对于任意一个状态谓词 $s \in \Sigma$,依赖表定义如式(2)所示:

$$D(s) = \{op \in A | s \in \text{Eff}(op)\} \quad (2)$$

即所有能够实现状态 s 的候选操作集合。依赖表因此提供了“哪些操作能够建立某一状态”的知识,为状态的依赖补全过程提供依据。

(2) 基于 ECA 规则的初始化机制

为确保 EnvAct 能够在冷启动阶段识别物理环境中的操作限制与状态影响关系,系统利用 LLM 从开源社区 ECA (Event-Condition-Action) 规则集中进行知识抽取。本文选取 Home Assistant 社区积累的蓝图 (blueprints) 脚本作为环境表征初始化的先验知识源。这些脚本来源于真实物联网自动化场景,包含大量设备操作、状态条件与触发事件之间的经验性关联,可为约束表和依赖表的初始化提供候选知识。具体而言,初始化过程围绕两类结构化知识展开:一类是面向设备操作的前置状态约束,用于初始化约束表;另一类是面向环境状态的候选影响操作,用于初始化依赖表。

环境表征的初始化与设备注册过程同步进行。对于新接入的设备或操作,系统首先解析其操作接口、作用对象和状态效果,并以注册操作为检索入口,从 Home Assistant 蓝图脚本中查找以该操作为 Action 的候选 ECA 规则。由于社区蓝图中包含大量用户个性化配置,系统不会直接将规则中的 Condition 项

写入约束表,而是将其视为候选前置约束,并调用 LLM 结合当前环境中已部署设备、可观测状态和物理常识进行筛选。对于具有普适意义的前置状态约束,系统将其归纳为约束表中的 $\text{Pre}(\text{op})$ 集合;对于仅反映用户偏好、时间习惯或特定场景配置的条件,则予以过滤。

系统检索到 ECA 规则: $\{\text{Event: Temperature} > 28\text{ }^{\circ}\text{C}, \text{Condition: Window} = \text{Closed}, \text{Action: turn_on_ac}\}$ 时, LLM 判断其中“窗户关闭”并非该自动化任务的偶然条件,而是保证空调制冷效能的物理前置约束,故抽取为: $\text{Pre}(\text{turn_on_ac}) = \{\text{Window} = \text{Closed}\}$ 。该约束可用于后续任务执行阶段的前置条件校验,当用户请求开启空调而当前窗户处于开启状态时,系统能够先补全“关闭窗户”操作,再执行空调控制。

对于新注册的感知状态或环境属性,系统则以该状态谓词为检索入口,从 Home Assistant 蓝图脚本中查找包含该状态的 Event 或 Condition 规则,并收集其对应的 Action 作为候选影响操作。需要说明的是, ECA 规则中的状态条件与动作之间仅表示经验性共现关系,并不必然构成直接因果关系。因此,系统同样调用 LLM 对候选动作进行语义判别,判断其是否能够建立、改变或维持该状态。若判断成立,则将该动作写入依赖表,形成状态到操作的逆向映射,否则将其丢弃。

例如,针对状态谓词 $\text{IndoorBrightness} = \text{Bright}$,系

统可检索到两类涉及室内亮度的规则:

(1) $\{\text{Event: Ambient_Light} < 20\text{ lx}, \text{Condition: Period} = \text{“Night”}, \text{Action: turn_on_light}\}$;

(2) $\{\text{Event: Ambient_Light} < 20\text{ lx}, \text{Condition: Period} = \text{“Day”}, \text{Action: open_curtain}\}$ 。

LLM 判断“开启灯带”和“打开窗帘”均可作为提升室内亮度的有效操作,但其适用条件分别与夜间和白天环境相关。因此,系统构建依赖 $D(\text{IndoorBrightness} = \text{Bright}) = \{\text{turn_on_light}, \text{open_curtain}\}$ 并在后续任务执行中结合当前环境状态选择合适的补全操作。

通过上述过程, EnvAct 能够在冷启动阶段从社区自动化规则中抽取第一版约束表与依赖表,同时借助 LLM 的语义判别能力过滤个性化规则、偶然触发条件和环境不适配规则,降低错误知识写入环境表征的风险。

为进一步直观展示上述冷启动过程,表 1 以“高温环境下开启空调”为例,给出了系统在设备操作与状态谓词逐项注册过程中生成约束表与依赖表的示例。

由表 1 可见,约束表用于刻画操作执行前需要满足的前置状态,依赖表则用于描述目标状态与候选操作之间的逆向映射。当合规器查询依赖表发现 turn_on_ac 的前置状态 $\text{Window} = \text{Closed}$ 未满足时,即可进一步查询依赖表,补全 close_window 操作,生成满足物理约束的执行序列。

表 1 约束表与依赖表的冷启动初始化示例

Table 1 Cold-start initialization example of the constraint table and dependency table

初始化阶段	示例说明
操作注册	当空调控制能力接入系统时,记录“开启空调”这一可调用操作
候选规则筛选	从预处理后的 Home Assistant 蓝图规则集中筛选与空调控制相关的候选 ECA 规则
环境适配过滤	若候选规则涉及的温度、窗户状态等条件在当前环境中可观测,则保留该规则;否则丢弃
约束归纳	结合 LLM 进行语义判别,将“窗户关闭”识别为开启空调前具有普适意义的前置状态约束
约束表生成	约束表项: $\text{Pre}(\text{turn_on_ac}) = \{\text{Window} = \text{Closed}\}$
状态谓词注册	当窗户开闭状态作为可感知属性接入系统时,记录状态谓词 $\text{Window} = \text{Closed}$
候选操作筛选	以 $\text{Window} = \text{Closed}$ 为目标状态,从候选规则集中筛选能够建立该状态的候选操作
依赖归纳	结合 LLM 进行语义判别,将“关闭窗户”识别为能够建立 $\text{Window} = \text{Closed}$ 的操作
依赖表生成	依赖表项: $D(\text{Window} = \text{Closed}) = \{\text{close_window}\}$
初始化结果	随着操作与状态谓词逐项注册,系统逐步形成第一版约束表与依赖表,为合规器的前置条件校验、依赖补全和执行校正提供支撑

2.1.3 动态演化与环境表征增量更新

针对物联网环境中设备能力、操作接口、状态属性及执行反馈不断变化的特点, EnvAct 支持环境表征的增量更新。需要说明的是,本文中的环境表征以图数据库形式组织,动态更新并非重新构建完整环境模型,而是对设备节点、状态属性、约束表项和依赖表项进行增量写入与修正。其更新过程主要包括注册

驱动的环境表征扩展和反馈驱动的环境表征修正两类路径。

当新设备、新操作或新状态属性接入系统时,环境状态引擎首先解析其设备类型、部署空间、操作接口与状态属性,并将相关设备节点、操作接口和状态谓词注册到环境表征中。随后,环境状态引擎触发候选规则检索流程,从预处理后的 Home Assistant 蓝图

规则集中筛选与当前设备能力或状态属性相关的候选 ECA 规则,并根据当前环境中状态是否可观测、操作是否可执行、空间对象是否匹配等条件进行环境适配过滤。

在此基础上,系统调用 LLM 对候选规则中的条件项与动作项进行语义判别,识别其中具有普适意义的前置状态约束和状态影响操作;环境状态引擎则根据判别结果,将通过筛选的约束项或依赖项增量写入约束表和依赖表。注册驱动的环境表征扩展流程如图 2 所示。

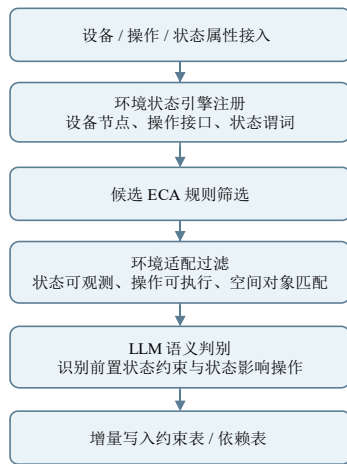


图 2 注册驱动的环境表征扩展流程

Figure 2 Registration-driven environmental representation expansion process

注册驱动路径主要用于处理设备、操作和状态属性接入带来的结构性变化,而反馈驱动路径则用于处理任务执行失败或反馈异常后的经验性修正。当任务执行失败时,反思器会分析执行日志、执行轨迹、环境快照和失败反馈,判断失败是否由前置约束缺失、依赖补全不足或延迟参数偏差引起。若失败原因指向前置约束缺失,则生成约束表修正建议;若指向依赖补全不足,则生成依赖表修正建议;若指向状态生效延迟不匹配,则生成延迟参数更新建议。上述修正结果经确认后写回环境表征,用于后续任务执行。反馈驱动的环境表征修正流程如图 3 所示。

2.1.4 运行支撑与感知执行

环境状态引擎负责维护环境表征与物理环境之间的一致性。它通过周期性采集传感器数据或接收设备反馈,更新环境表征中的节点状态与属性值,使系统能够实时感知空间、设备与环境属性的变化。同时,环境状态引擎负责将上层模块生成的操作指令下发至设备代理层,并接收执行反馈,用于后续状态校验与异常标注。当数据存在冲突或缺失时,系统会对相关状态进行标记或修正,以保证环境表征与物理环

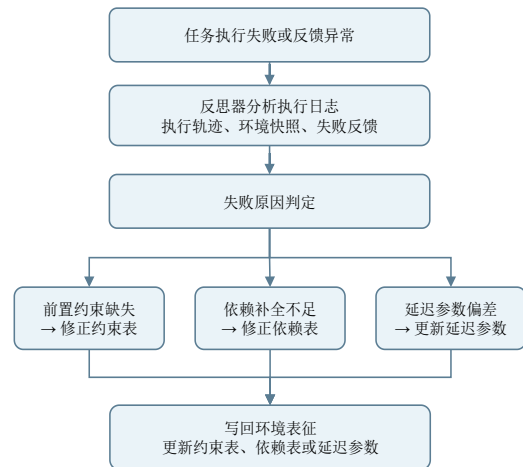


图 3 反馈驱动的环境表征修正流程

Figure 3 Feedback-driven environmental representation correction process

境的一致性。

设备代理作为系统与异构设备之间的适配层,负责将控制指令映射到具体的设备通信协议与执行接口。为了提升可扩展性与易用性,本研究采用 KubeEdge 的 Mapper 框架作为设备代理的实现方式。Mapper 提供了对多种工业与物联网协议(如 MQTT、Modbus、OPC-UA 等)的支持,使系统能够以插件化方式接入不同类型的设备。通过设备代理,系统实现了对多源异构设备的统一管理与灵活接入。

2.2 规划器

规划器作为系统的起始模块,负责将用户的自然语言需求转化为可执行的操作序列。在大模型主导的智能控制方法中(如 Sasha 等),用户意图解析通常依赖单次输入的语义信息,容易忽略环境状态与历史交互的影响,从而导致生成结果与实际场景不匹配。为此,本文在大模型生成框架的基础上,引入了环境检索增强与会话记录融合两种机制,使得规划器能够在操作生成阶段同时感知环境条件与用户偏好,从而提升结果的合理性与上下文一致性。

规划器的工作流程主要包括三个阶段。

(1) 用户需求解析与环境信息收集

当用户提出需求时,规划器首先通过大模型判断命令中涉及的位置信息,并根据这些位置信息从环境表征层检索相关的环境状态和设备接口描述。同时,系统从会话记录的向量数据库中检索与用户偏好相关的信息,并进行偏好识别与语义解析。通过关键词(如“喜欢”“不要”“必须”等)筛选会话记录中的相关语句,系统再借助大语言模型进行约束归纳,汇总与用户个性化偏好相关的内容,将其作为辅助上下文供规划器参考。这些信息共同构成了大模型生成操作

序列的完整上下文。

(2) 提示词生成与上下文整合

为系统化表示提示词内容, 本文将提示词形式化为一个六元组, 如式(3)所示:

$$\text{Prompt} = (T, P, E, I, R, F) \quad (3)$$

其中, T (Task)表示任务信息, 用于描述规划目标与执行原则; P (Preference)表示用户偏好, 用于刻画用户对

任务执行的个性化需求; E (EnvState)表示环境信息, 由接口返回的实时环境状态构成; I (InterfaceDesc)表示设备接口描述, 用于定义可调用的设备操作及参数; R (Request)表示用户需求, 即用户输入的自然语言任务命令; F (FormatInfo)表示输出格式, 用于规范规划结果的结构化形式。规划器基于上述信息生成结构化输出, 其过程如图4所示。

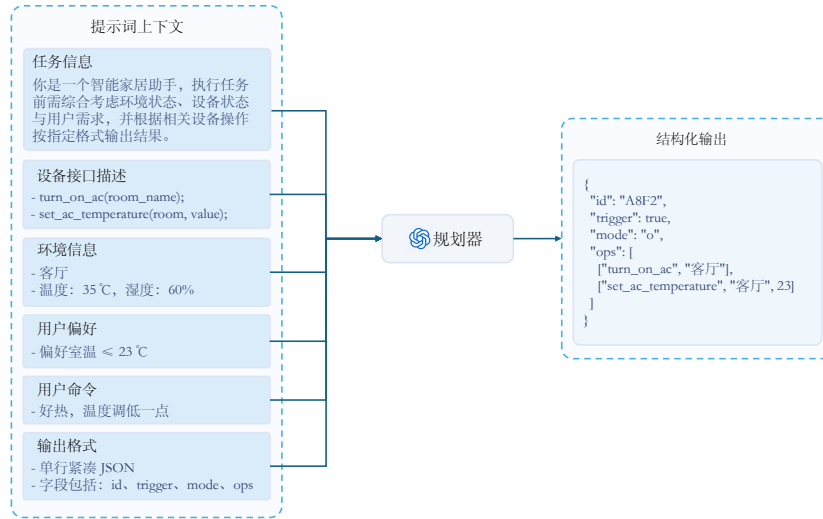


图4 规划器基于提示词上下文生成结构化输出的示意图

Figure 4 Schematic diagram of the planner generating structured output based on prompt context

(3) 操作序列生成与协议封装

规划器将任务信息、用户偏好、环境信息、设备接口描述、格式约束和用户需求一并作为上下文输入大模型。模型在综合推理后, 输出符合协议规范的紧凑型结构化响应。

我们将模型输出的响应定义为一个四元组, 如式(4)所示:

$$\text{Output}(\text{id}, \text{trigger}, \text{mode}, \text{OpSeq}) \quad (4)$$

其中: id 为长度为4的十六进制随机字符串, 用于作为指令的唯一标识符, 并支持接收端去重处理, 确保在TCP重传机制下指令仅被执行一次; trigger 表示触发条件语义, 当指令由用户即时下达时取值为true, 当指令由环境自动联动触发时表示为特定逻辑表达式, 例如 $\text{LivingRoom.Temperature} > 35^\circ\text{C}$; mode 表示指令时效模式, 其中 o (once)表示单次脉冲式执行, p (persistent)表示持久性的状态维持; OpSeq 表示有序操作序列, 可进一步定义为

$$\text{OpSeq} = [\text{op}_1, \text{op}_2, \dots, \text{op}_n],$$

$$\text{op}_i = (\text{action}_i, \{\text{param}_1, \text{param}_2, \dots, \text{param}_{m_i}\}) \quad (5)$$

其中, op_i 表示第 i 个操作; action_i 表示操作名称, $\{\text{param}_1, \text{param}_2, \dots, \text{param}_{m_i}\}$ 表示该操作对应的参数

集合。

2.3 执行器

执行器是 EnvAct 框架中的核心控制中枢, 负责管理任务从“条件触发”到“物理执行”的全生命周期。不同于传统的静态指令下发, 执行器与规划器、合规器紧密协同, 实现了基于环境反馈的闭环控制。其工作逻辑分为以下四个阶段。

(1) 触发监测与任务激活

执行器接收规划器生成的触发条件与初始操作序列。在此阶段, 执行器进入实时监听模式, 持续监测环境表征层中的状态变迁。

条件匹配。当且仅当环境状态满足规划器定义的触发逻辑(如 $\text{livingroom, temp} > 28$ 等), 执行器激活当前任务。

合规处理。任务激活后, 执行器并不直接操作设备, 而是将初始序列移交给合规器, 由其基于约束表与依赖表进行安全性校验与补全。

(2) 基于偏序图的并行调度

执行器接收合规器补全后的操作偏序图 G 。为了实现高效且合规的执行, 执行器运行算法1对操作进行动态调度。

就绪集动态选择。算法 1 在任一时刻扫描 G , 若操作 v 的所有前驱节点 (即偏序图中的父节点 $\text{Pred}(v)$) 已完成物理反馈, 则将 v 纳入就绪集 (ReadySet)。

并行化策略。执行器根据底层设备的并发处理能力, 从就绪集中选择互不冲突的操作子集并行下发指令。通过并行调度, 系统在确保因果顺序 (即满足 $\text{Pred}(v)$ 完成) 的前提下, 显著降低了总任务执行时间。

(3) 设备调用与闭环验证

在具体指令下发阶段, 执行器实现了针对物理不确定性的闭环控制机制。

指令映射与超时重试。执行器将操作 v 映射为底层物理指令, 并设定重试次数上限与超时时间。若设备调用失败, 执行器将自动触发重试逻辑。

基于延迟的效果验证。在操作下发后的设定延迟 $\text{Latency}(v)$ 结束后, 执行器检索环境表征层以验证 $\text{Eff}(v)$ 是否达成。

异常处理与重规划。若执行失败 (如重试后仍未满足约束或发生物理冲突), 执行器立即停止当前序列, 并将环境快照与失败上下文移交给规划器。由规划器在新的环境状态下重新生成修复序列并下发。

(4) 结果输出与反思支持

当操作序列成功执行完毕, 执行器将完整的执行轨迹 L (包含开始/结束时间、传感器反馈、环境变迁等) 输出给反思器。这为反思器提取用户偏好、更新初始化阶段构建的约束表与依赖表提供核心数据支持。

算法 1 执行器的并行调度与验证算法

输入: 偏序图 $G=(V, E)$, 设备代理映射 $\text{Map}()$

输出: 执行轨迹 L , 记录每个操作的执行状态、起止时间、重试次数与验证结果

1. 初始化: $S \leftarrow \emptyset, D \leftarrow \emptyset, L \leftarrow \emptyset$;
2. $Q \leftarrow \{v \in V \mid \text{Pred}(v) = \emptyset\}$;
3. 当 $|D| < |V|$ 时
4. 选择并发批次 $B \leftarrow \text{SelectBatch}(Q)$;
5. 并行对每个 $v \in B$ 执行:
6. 若 $\text{Eff}(v)$ 已满足, 则记为 success;
7. 否则, 调用 $\text{RunAndVerify}(v)$ 执行并验证操作;
8. 根据执行结果更新 S, D 和 L ;
9. 结束并行
10. $Q \leftarrow \{u \in V \setminus D \mid \text{Pred}(u) \subseteq S\}$;
11. 返回 L

为将上述执行逻辑转化为可工程化的调度流程, 执行器引入并行调度与验证算法, 如算法 1 所示。该算法以合规器生成的偏序图 $G=(V, E)$ 为输入, 其中 V 表示待执行操作集合, E 表示操作之间的依赖关系。算法中, S 表示成功集合, D 表示已决策集合, Q 表示

当前满足前驱约束的就绪集合; $\text{RunAndVerify}(v)$ 封装设备调用、状态查询、超时等待、延迟验证与重试过程。执行结束后, 执行器将完整执行轨迹 L 输出给反思器, 为其提取用户偏好、更新约束表与依赖表提供数据支持。

2.4 合规器

规划器生成的初始操作序列通常只反映用户需求对应的目标动作, 可能尚未充分考虑当前环境状态约束。例如, 在执行“空调打开操作”之前, 若当前窗户并未关闭, 则该操作的前置条件尚未满足。为此, 合规器需要基于环境表征层提供的约束表和依赖表, 对初始操作序列进行补全与重排, 将其转化为满足环境约束和操作依赖关系的偏序执行图。具体而言, 约束表 $C(\cdot)$ 用于给出操作执行前所需满足的前置状态集合 $\text{Pred}(a)$, 依赖表 $D(\cdot)$ 用于根据未满足状态查找能够产生该状态的候选修复操作。整个偏序图生成过程如算法 2 所示。

算法 2 基于约束表和依赖表的偏序图生成算法

输入: 初始环境状态 S_0 ; 初始动作序列 $I=[a_1, a_2, \dots, a_m]$; 约束表 $C(\cdot)$; 依赖表 $D(\cdot)$

输出: 偏序执行图 $G=(V, E)$

1. 初始化: $V \leftarrow \emptyset; E \leftarrow \emptyset; S \leftarrow S_0$
 2. $\text{done} \leftarrow \emptyset, \text{stack} \leftarrow \emptyset$;
 3. 对于每个 $a_i \in I$ 执行:
 4. 若 $\text{INSERT}(a_i) = \text{false}$, 则返回前置条件补全失败错误;
 5. 返回 $G=(V, E)$ 。
- 过程 $\text{INSERT}(a)$
6. 若 $a \in \text{done}$ 中, 则返回 true
 7. 若 $a \in \text{stack}$, 则返回循环依赖错误;
 8. 将 a 加入 stack ;
 9. 对于每个 $p \in \text{Pre}(a)$ 执行:
 10. 若 $\text{Sat}(S, p) = \text{false}$, 则
 11. $R \leftarrow D(p)$;
 12. 若 $R = \emptyset$, 则返回 false;
 13. 选择修复操作 $r \in R$
 14. 若 $\text{INSERT}(r) = \text{false}$, 则返回 false;
 15. $E \leftarrow E \cup \{(r, a)\}$;
 16. 若 $a \notin V$, 则 $V \leftarrow V \cup \{a\}$;
 17. $S \leftarrow \text{Apply}(S, a)$;
 18. $\text{done} \leftarrow \text{done} \cup \{a\}, \text{stack} \leftarrow \text{stack} \setminus \{a\}$;
 19. 返回 true

算法 2 中, $\text{Sat}(S, p)$ 用于判断当前预测环境状态 S 是否满足前置条件 p ; 当 $\text{Sat}(S, p) = \text{false}$, 算法通过依赖表 $D(p)$ 查找可补全该状态的修复操作 r , 并递归调用 $\text{INSERT}(r)$ 将其插入到当前操作 a 之前。若修复操作成功插入, 则在边集合 E 中加入依赖边 (r, a) , 表示 r

需要先于 a 执行;若不存在可用修复操作,或递归过程中出现循环依赖,则返回前置条件补全失败错误。`done` 用于记录已经完成插入的操作, `stack` 用于检测当前递归链中的循环依赖。`Apply( $S, a$ )` 根据操作 a 的效果更新预测环境状态,从而支持后续操作前置条件的判断。最终生成的偏序执行图 $G=(V, E)$ 中, V 表示操作节点集合, E 表示操作之间的先后依赖关系,可进一步用于拓扑排序和并行调度。

2.5 反思器

反思器是 EnvAct 框架中的反馈优化模块,主要负责接收执行器返回的执行结果,并结合执行日志、环境快照、设备反馈和用户反馈对系统运行过程进行分析。与规划器、合规器和执行器直接参与当前任务执行不同,反思器主要作用于任务完成后的经验沉淀与环境表征修正,其目标是将一次任务执行过程中的成功经验、失败原因或用户偏好转化为后续任务可复用的结构化信息。

当任务执行成功时,反思器将执行过程、操作序列、环境状态变化、用户反馈和评估结果整合至会话日志中,并采用 `text-embedding-v1` 嵌入模型进行向量化处理,随后存储于向量数据库。该过程使系统能够在后续任务规划时检索历史交互记录,结合用户偏好、常用设备选择和既往执行模式生成更加符合上下文的操作方案。例如,用户对执行结果的满意反馈以及在相似场景下重复选择的设备或控制方式,均可作为偏好信息被写入历史记录,并在后续规划阶段作为辅助上下文供规划器参考。

当任务执行失败或用户对执行结果提出否定反馈时,反思器进一步分析执行日志、环境快照、失败反馈、用户反馈以及合规器生成执行计划时涉及的约束表或依赖表项,并将其共同构造提示词,调用 LLM 判断失败是否与环境表征不完整、用户偏好未被满足或执行参数设置不合理有关。若失败原因指向前置约束缺失,反思器生成约束表修正建议;若失败原因指向依赖补全不足,则生成依赖表修正建议;若失败原因与状态生效延迟不匹配有关,则对相应操作的延迟参数进行更新。上述修正结果按照图 3 所示

的反馈驱动修正路径写回环境表征,使系统能够在后续面对相似任务时避免重复发生相同错误。

3 实验评估

我们使用 Python 3.11.9 作为编程语言实现了 EnvAct 系统。环境表征数据存储在 Neo4j 5.26.10 数据库中,以图形数据库形式组织环境的各类信息。会话记录通过 `text-embedding-v1` 技术进行处理,并存储到向量数据库 Chroma 1.1.0 中,以便进行高效的语义搜索和相似度匹配。为了评估 EnvAct,我们开展了一系列实验研究,以回答以下研究问题(Research Questions, RQ)。

RQ1:相较于基准方法(SAGE, Sasha), EnvAct 在处理物联网多样化任务(如意图解析、设备消歧、持久性任务等)时的端到端任务执行效能如何?

RQ2:相较于基准方法(SAGE, Sasha), EnvAct 在推理延迟与 Token 消耗等系统开销方面表现如何?其性能增益是否具有成本效益?

RQ3: EnvAct 的关键模块与闭环校正机制在提升执行可靠性方面发挥了何种作用?

RQ4:在物理沙盘环境实验中, EnvAct 系统的可用性、适应性与实时性表现如何?

3.1 实验设置

本研究实验分为两个部分:第一部分基于任务数据集测试,第二部分基于沙盘环境测试。在这一节中,我们将详细介绍数据集测试的设置与方法,沙盘环境的搭建内容将在后续部分展开。

3.1.1 静态任务数据集设计

为了全面评估 EnvAct 在智能家居场景下的任务规划与执行能力,并确保评估维度的完备性,我们参考了该领域近期代表性研究 SAGE^[9]提出的任务分类法。这种分类方式基于智能家居任务中面临的挑战类型,对任务场景进行系统化划分,如表 2 所示。

3.1.2 动态执行干扰实验设计

为了定量评估系统 EnvAct 在复杂物联网环境中的可靠性,本研究构建了一套基于故障注入的动态仿真评估机制。该部分实验在基础的指令逻辑测试之

表 2 典型指令任务设计及对应操作

Table 2 Typical instruction task design and corresponding operations

类别	核心定义简述	典型任务实例
个性化	结合用户偏好习惯	“按我平时的习惯准备淋浴水温。”
意图解析	处理模糊/间接指令	“家里太闷了。”
设备消歧	定位正确的设备	“关掉书桌左边那盏灯。”
持久性	长期监测与条件触发	“如果晚上十点我还没关电视,请提醒我。”
链式操作	复杂多步骤指令分解	“我要开会了,把百叶窗关上并开启降噪模式。”
直接指令	简单明确的控制	“打开客厅空调。”

外,引入了执行期的闭环反馈考察。在维度的选择上,本研究遵循物联网交互全链路覆盖的原则,旨在还原系统从指令传输、设备执行到外部环境交互的全生命周期中可能遭遇的关键失效模式。我们将物理环境中的不确定因素抽象为以下四个空间维度,用以压力测试系统的自愈与纠偏能力:

随机通信延迟,对应通信层扰动。仿真程序在指令下发过程注入随机时耗或模拟数据包丢失,表征异构网络中常见的信道拥塞或通信中断现象,用以测试系统对设备执行超时的容错能力。

执行端点异常,对应执行层扰动。采用概率模型模拟执行设备瞬时失效或传感器数据漂移的情况,旨在验证系统在面对执行反馈异常时的自愈与重规划能力。

异步状态冲突,对应交互层扰动。模拟现实中典型的“人机共存”场景,即在系统执行链条中随机插入由外部干预引起的状态变更(如门窗误关导致扫地机器人路径受阻),考察系统对设备执行环境约束变化的实时识别。

环境语义漂移,对应语义层扰动。模拟物理环境参数(如气温、光照)的剧烈变化,导致原定“持久性任务”与当前环境约束不再匹配的场景(如气温骤降导致的降温相关指令不再适用),验证系统基于实时环境感知的策略调整能力。

这一部分实验将重点展示 EnvAct 的动态反馈与合规校正机制如何捕捉上述非理想化反馈,并通过动态调整设备选择或操作执行顺序,确保高层任务目标的最终达成。

3.1.3 评价指标与比较对象

为了确保评价的客观性与一致性,我们采用响应时间、可靠性和任务成功率(Success Rate)作为核心指标。

响应时间是指从用户发出请求到任务完成的总时间。响应时间反映了系统在执行任务时的实时性和效率,直接影响系统的稳定性和用户体验。

可靠性是指在动态且不确定的物理环境中,系统在给定任务集合上,能够持续产生可执行且语义一致的控制行为,在满足环境约束的前提下稳定实现用户定义的目标的概率。其计算公式如式(6)所示:

$$\text{可靠性} = \frac{\text{语义一致且满足环境约束的任务数量}}{\text{总任务数}} \times 100\% \quad (6)$$

判定“可靠”的条件包括:

(1)控制行为的逻辑连贯性。在长序列任务或多设备联动场景下,系统生成的指令序列不存在前后矛盾。

(2)环境约束的强制遵守。在动态变化的物理环境中,系统必须始终将“安全性”作为最高优先级。在完成用户目标的过程中,绝不触碰硬性约束红线。

(3)异常干扰下的状态恢复能力。当传感器数据出现波动、物理反馈延迟、设备异常情况时,系统能够通过重试、修正或再规划来维持运行状态。

任务成功率是指在一定时间内成功完成的任务数量与总任务数量之比。任务成功率反映了系统在任务完成时是否达到用户预期的终点。其公式如式(7)所示:

$$\text{成功率} = \frac{\text{成功完成的任务数量}}{\text{总任务数}} \times 100\% \quad (7)$$

判定“成功”的条件包括:

(1)意图解析正确。系统能够准确理解用户需求并生成合理操作序列。

(2)任务完成性。动作序列能够在给定的资源与时序限制下完成预期目标。

(3)环境效果达成。执行序列完成后,环境状态需符合用户预期的状态目标。

为了全面评估 EnvAct 系统在不同任务类型下的性能,我们选择了 qwen-plus 和 GPT-4o 作为大语言模型,并与现有的两个基准模型 Sasha^[13]和 SAGE^[9]进行对比。我们的对比重点是任务成功率,该指标衡量系统在执行任务过程中成功完成的比例,能够反映出模型的有效性和实际执行能力。

3.2 RQ1:端到端任务执行效能分析

为了回答 RQ1,本文从响应时间和任务成功率两个方面评估 EnvAct 的端到端任务执行效能。响应时间反映系统处理用户请求的效率,任务成功率反映系统在不同物联网任务中完成用户目标的能力。通过与 Sasha 和 SAGE 进行对比,本文进一步分析结构化环境表征对任务执行效率与执行效果的影响。

3.2.1 响应时间

EnvAct 的整体框架通过三个核心模块支持用户交互:规划器、合规器和执行器。规划器负责生成操作计划,合规器确保操作的合法性,执行器则执行具体操作并进行效果验证。反思器在后台异步执行,用于收集执行结果并进一步优化后续操作。由于反思器是在操作完成后启动的,且不与用户交互过程直接关联,因此不会影响系统的响应时间。同时,环境状态的表征由环境状态引擎在后台持续更新,它通过周期性采集传感器数据或接收设备反馈,确保系统实时感知环境变化。因为环境状态更新与任务执行并行进行,它也不会影响系统对用户的响应时间。

如表 3 所示, qwen-plus 模型响应时间普遍优于 GPT-4o 模型,两种不同大模型之间的响应时间差异

是由不同大模型的实现架构和不同网络条件导致的通信时延引起的。从不同系统响应时间的对比维度来看,EnvAct 优于 Sasha 和 SAGE,这一优势主要源于各系统架构设计所导致的大模型调用次数差异。具体而言,Sasha 将系统划分为 Clarify、Filter、Plan、Implement 和 Feedback 五个模块,其中前四个模块均依赖于大模型调用,因此在单次执行过程中至少需调用四次大模型;SAGE 则采用 ReAct 架构,对于涉及多设备操作的复杂任务,其需逐步推理并执行,从而引发较高的推理时间开销;相比之下,EnvAct 通过环境表征机制一次性推理生成执行方案,并在后续执行阶段仅需进行合规化处理,从而能够更稳定地应对复杂任务,避免因错误引发的延迟,在效率与可靠性方面实现更佳的平衡。

表 3 各系统平均响应时间				单位:s
Table 3 Average response time of each system				unit: s
大模型	Sasha	SAGE	EnvAct	
qwen-plus	7.33	8.32	4.79	
GPT-4o	7.39	8.69	5.95	

从表 4 可以看出,EnvAct 的响应时间主要由三个关键模块规划器、合规器和执行器的响应时间构成。整体来看,规划器占据了系统响应时间的最大份额,无论是在 qwen-plus 模型还是 GPT-4o 模型下,规划器的响应时间均占总响应时间的 85% 以上,尤其是在使用 GPT-4o 模型时,占比为 90.06%。这一结果表明,系统的大部分时间主要消耗在生成任务计划和推理过程上,即大模型的推理部分。

表 4 EnvAct 各组件响应时间百分比				单位:%
Table 4 Response time percentage of each EnvAct component				unit: %
大模型	规划器百分比	合规器百分比	执行器百分比	
qwen-plus	86.09	3.01	10.9	
GPT-4o	90.06	1.96	7.98	

相比之下,执行器的响应时间较短,因为我们采用的是基于数据集的测试,不需要真正等待设备执行完成。合规器的响应时间则相对稳定,起到的作用是确保设备操作的合理性和合规性,处理时间相对较短。通过后台组件的持续运行,EnvAct 能够在任务执行的过程中并行处理环境状态更新与结果反思,从而确保系统能够实时感知环境变化,而这一过程不会影响系统的响应时间。

整体而言,EnvAct 在响应时间上表现良好,能够有效地平衡任务执行效率与系统的稳定性,进一步提高了其在动态环境中处理复杂任务的能力和可靠性。

3.2.2 任务成功率

图 5 展示了不同系统在总体任务集上的任务成

功率。结果显示,EnvAct 在 qwen-plus 与 GPT-4o 两种大模型驱动下均显著优于基线系统 Sasha 和 SAGE。具体而言,在 qwen-plus 支持下,EnvAct 的成功率达到 88.3%,较 Sasha(43.3%)提升 45.0 个百分点,较 SAGE(75.0%)提升约 13.3 个百分点;在 GPT-4o 支持下,EnvAct 的成功率达到 90.0%,分别比 Sasha 和 SAGE 高出 48.3 个百分点和 13.3 个百分点。

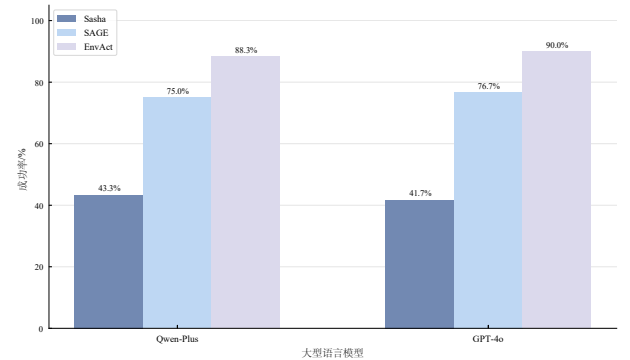


图 5 不同系统在总体任务集上的任务成功率比较
Figure 5 Comparison of task success rates among different systems on the overall task set

这一优势主要得益于结构化环境表征与执行前合规校正机制。二者能够减少环境依赖遗漏、设备选择错误和操作顺序不合理等问题,从而提升复杂物联网任务的端到端执行能力。不过,少量失败仍出现在两类场景中:一是用户指令较为模糊或依赖隐含上下文时,模型可能误判用户真实意图;二是任务涉及安全敏感设备联动时,系统可能受限于当前领域知识覆盖范围,无法完全识别潜在风险。因此,后续仍需进一步引入领域知识约束与安全校验机制。

3.3 RQ2:推理效率与计算开销分析

为了评估 EnvAct 框架在实际部署中的资源消耗及成本效益,本节对比了 EnvAct 与基准方法(Sasha, SAGE)在推理延迟与 Token 消耗方面的表现。

3.3.1 推理时延分析

系统的推理时延直接影响物联网任务的响应速度。如图 6 所示,在推理时延维度,EnvAct 表现出显著的优势。SAGE 由于采用 ReAct 架构,在处理复杂任务时需要多轮“思考-行动-观察”的迭代,导致推理路径较长,其时延在所有系统中最高。Sasha 虽然通过流水线处理任务,但在单次任务中需要依次调用五个模块,其中四个模块(Clarify、Filter、Plan、Implement)涉及大模型推理,多次的网络往返累积了较高的通讯与推理时延。

相比之下,EnvAct 采用了“规划-执行-合规”的一次性推理机制,将实时环境状态、用户偏好与设备接

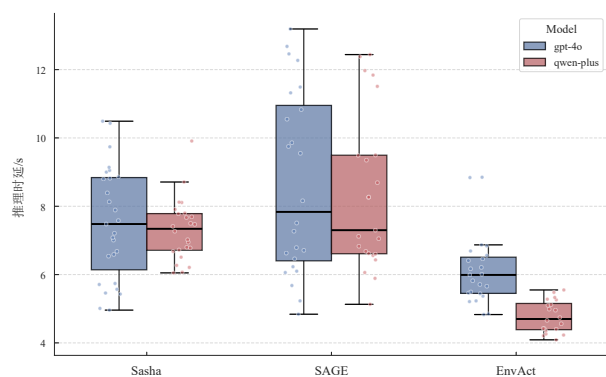


图6 不同系统的推理时延比较

Figure 6 Comparison of inference latency among different systems

口信息深度对齐后统一输入模型。实验数据表明,在使用 qwen-plus 模型时,EnvAct 的推理时延中位数约为 5 s,显著低于 Sasha 的 7 s 和 SAGE 的 8~9 s。这进一步验证了结构化的环境表征能有效精简大模型的逻辑推理链条,显著提升系统的执行效率。

3.3.2 Token 消耗分析

在 Token 消耗方面,如图 7 所示,各系统呈现出明显差异。SAGE 采用 ReAct 式交互机制,推理、工具调用和环境反馈会随交互轮数不断累积到上下文中,因此其 Token 消耗随任务复杂度波动较大,并在复杂任务中达到 10 000 ~ 20 000 的高位区间。Sasha 的 Token 消耗最低,因为其单次调用传递的信息较为精简,整体维持在 2 000 左右,但这种轻量化以牺牲任务成功率为代价,难以支撑复杂应用。相比之下,EnvAct 的 Token 消耗位于中间区间,主要集中在 5 000 ~ 6 000 左右。尽管其提示词中引入了结构化环境表征,增加了一定基础开销,但由于减少了多轮工具交互与上下文累积,其总体开销明显低于 SAGE,并在成功率和资源消耗之间取得了更好的平衡。因此,EnvAct 在保持较高任务成功率的同时,将推理时延和 Token 消耗控制在相对合理的范围内,体现出较高的资源效率与部署可行性。

3.4 RQ3: 关键模块与执行可靠性分析

为回答 RQ3,本文围绕关键模块贡献与动态环境可靠性展开评估。一方面,通过合规器和反思器消融实验分析各模块对任务成功率和持续优化能力的影响;另一方面,通过故障注入方式模拟通信延迟、执行端点异常、异步状态冲突和环境语义漂移等扰动,考察完整 EnvAct 在非理想物理环境下的执行可靠性与恢复能力。

3.4.1 合规器消融实验

为验证合规器在约束校验与操作补全中的作用,本文设计了合规器消融实验,比较完整 EnvAct 与去

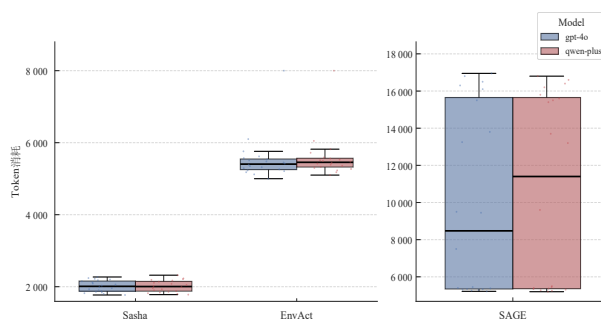


图7 不同系统在任务执行过程中的 Token 消耗比较

Figure 7 Comparison of token consumption among different systems during task execution

除合规器后的 EnvAct 变体在任务成功率上的差异。无合规器版本仍由规划器生成初始操作序列,并由执行器下发执行;反思器仅保留用户偏好和会话记录更新能力。由于缺少合规器对约束表和依赖表的调用,该版本不再具备执行前的前置条件校验、依赖补全和偏序执行图构建能力,因而主要依赖规划器一次性生成满足物理约束的完整操作计划。

如图 8 所示,完整 EnvAct 在 qwen-plus 和 GPT-4o 模型下的任务成功率分别达到 88.3% 和 90.0%;去除合规器后,成功率分别下降至 28.3% 和 33.3%,较完整系统下降 60.0 和 56.7 个百分点。这表明,仅依赖大语言模型生成操作序列,难以稳定地满足物联网环境中的物理约束和设备依赖关系,合规器是影响 EnvAct 任务执行成功率的关键模块。

造成这一差异的主要原因在于,规划器生成的操作序列虽然在语义上可能合理,但未必满足执行层面的前置状态要求。例如,在窗户处于开启状态时直接执行“开启空调”,或者在多设备联动任务中忽略设备间的先后依赖,都会导致任务失败。合规器能够在执行前查询约束表和依赖表,对初始操作序列进行校验与补全,并构建满足依赖关系的偏序执行图,从而将语义计划转化为符合物理环境约束的可执行计划。

因此,合规器不仅承担执行前检查功能,更是连接大模型语义规划与物理设备执行的重要中间层。消融结果说明,缺少合规器时,系统容易退化为“规划后直接执行”的开环模式;完整 EnvAct 则能够通过合规校正显著提升复杂物联网任务的执行正确性与稳定性。

3.4.2 反思器消融实验

为验证反思器在系统持续优化与闭环自适应中的作用,本文进一步设计了反思器消融实验。实验在保持规划器、合规器和执行器不变的情况下,比较完整 EnvAct 与去除反思器后的 EnvAct 变体在多轮评估

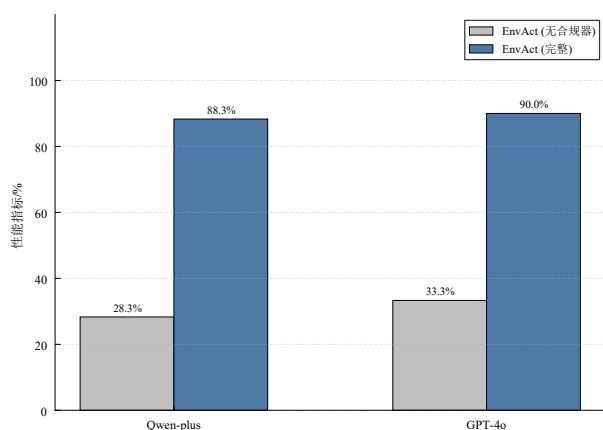


图8 合规器消融实验下的任务成功率比较

Figure 8 Comparison of task success rates in the ablation experiment of the compliance module

过程中的任务成功率变化。根据本文方法设计,反思器主要负责接收执行器的执行结果,并基于执行日志、用户反馈和环境状态变化,对会话记录、用户偏好以及约束表和依赖表中的潜在不合理项进行持续修正,从而使系统能够在后续任务中复用历史经验并优化执行策略。

如图9所示,在初始状态下,完整 EnvAct 与无反思器版本的任务成功率基本一致。其中,基于 GPT-4o 的两组系统初始成功率均为 81.7%,基于 qwen-plus 的两组系统初始成功率均为 80.0%。这一结果表明,在尚未积累执行反馈之前,反思器尚未对环境表征和历史经验产生修正作用,系统性能主要由规划器、合规器和执行器共同决定。

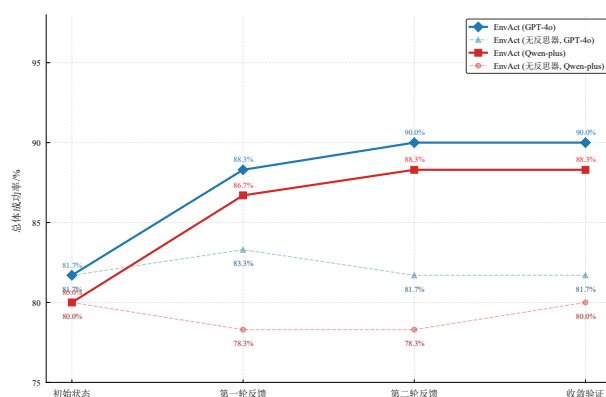


图9 反思器反馈更新对任务成功率的影响

Figure 9 Effect of reflector feedback updates on task success rate

随着反馈信息的积累,完整 EnvAct 的任务成功率呈现出稳定上升趋势。在 GPT-4o 模型下,完整 EnvAct 的成功率由初始状态的 81.7% 提升至第一轮反馈后的 88.3%,并在第二轮反馈后进一步提升至 90.0%,随后在收敛验证阶段保持稳定。在 qwen-plus

模型下,完整 EnvAct 的成功率也由初始状态的 80.0% 提升至第一轮反馈后的 86.7%,并在第二轮反馈后达到 88.3%,最终保持在该水平。这说明反思器能够有效利用执行过程中的失败案例、环境反馈和用户偏好信息,将其转化为可复用的结构化经验,从而减少后续任务中的重复性规划错误和不合理操作。

相比之下,去除反思器后的系统不具备环境表征更新和经验沉淀能力,因此其多轮评估结果并不代表真正的反馈迭代,而是在相同任务集上的重复评估。无反思器版本在 GPT-4o 下的成功率仅在 81.7% 至 83.3% 之间小幅波动,在 qwen-plus 下则在 78.3% 至 80.0% 之间变化。由于重复测试过程中未引入新的反思记录、偏好记忆或约束修正,上述波动主要来源于大语言模型输出随机性和任务执行不确定性。由此可见,缺少反思器时,系统难以将历史执行结果转化为后续任务可复用的结构化经验,因而无法形成稳定的持续性能增益。

3.4.3 动态扰动下的执行可靠性

在完成关键模块的消融分析后,本文进一步考察完整闭环机制在动态扰动环境中的可靠性表现。本研究构建了一套基于故障注入的动态仿真评估机制。实验重点考察系统从指令传输、设备执行到外部环境交互的全生命周期中,对关键失效模式的自愈与纠偏能力。如图10所示,实验记录了系统在四种典型扰动下的执行可靠性(柱状图)与平均恢复时间(折线图)。

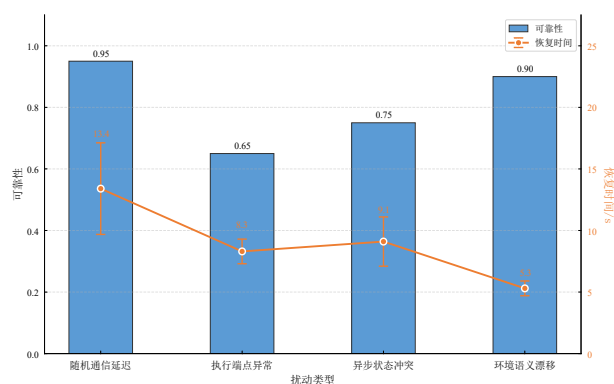


图10 EnvAct在四种扰动下的执行可靠性与平均恢复时间

Figure 10 Execution reliability and average recovery time of EnvAct under four types of disturbances

在随机通信延迟场景下,EnvAct 表现出极高的鲁棒性,可靠性接近 95%。虽然执行器内置的超时重试逻辑导致其恢复时间达到了最高值(约 13.5 s),但系统能通过指令重发有效对冲网络层丢包或拥塞的影响。

在执行端点异常场景下,可靠性降至约 65%。这是由于传感器数据漂移或设备瞬时失效往往会导致系统产生错误的语义判定。例如,当传感器反馈错误

的数值时,系统可能误判环境状态已达成,这种源头上的感知偏差使得合规器难以通过逻辑推导进行事后修正。

对异步状态冲突(如人机共存导致的突发障碍),系统的执行可靠性约为 75%。该场景的修复能力高度依赖于设备代理反馈信息的粒度,其瓶颈在于系统从粗粒度的设备反馈中难以精准解析故障的物理成因。若系统能通过环境状态的变化明确冲突诱因(如感知到窗户被手动关闭导致空调前置约束失效),则能通过合规器重新补全操作;反之,若设备仅反馈抽象的执行异常(如扫地机器人仅反馈“路径受阻”而非具体的“房门被关闭”),由于规划器无法获取故障背后的物理真相,其生成的修复序列往往无法精准解决底层障碍,导致任务最终失败。

在环境语义漂移场景下,系统可靠性保持在 90% 以上。面对气温突变等情况,原有的持久性任务(如基于旧温标的制冷需求)将不再匹配实时环境约束。EnvAct 能够快速捕捉参数漂移,快速上报用户进行二次确认,确保了执行决策在动态环境中的语义一致性。

综合上述压力测试结果,EnvAct 在动态干扰环境下的平均执行可靠性达到了 81.25%。实验结果证明,尽管在面对底层感知粒度受限及感知源头误判时仍存在改进空间,但 EnvAct 凭借“状态监测-合规校正-闭环重规划”的深度对齐机制,显著提升了物联网自动化执行在非理想物理环境下的容错能力与鲁棒性。

3.5 RQ4: 物理沙盘环境中的系统表现

为了进一步验证系统在真实物理环境下的可行性与稳定性,我们在仿真测试的基础上搭建了一个智能家居沙盘,并在该环境中对 EnvAct 的执行效果进行了验证。沙盘环境包括客厅、卧室与厨房区域,部署了空调、加湿器、智能灯、窗帘、传感器等多种设备,如图 11 所示。

3.5.1 沙盘搭建方法

沙盘系统整体架构由上位机、消息中间件和下位机三部分组成:

服务层(上位机)。基于 FastAPI 搭建 Web 服务,用于接收和处理来自用户或系统的 HTTP 请求。

消息中间件。采用 MQTT Broker 实现异步消息路由,保证系统各模块与设备之间的低延迟通信。

设备网关(下位机)。由 ESP32 节点构成,负责具体设备的控制与环境数据采集。

3.5.2 实验任务与场景

在沙盘环境中,我们设计并复现了多个关键任务场景,以全面考查系统在环境感知、依赖推理与多设



图 11 沙盘俯视图

Figure 11 Top view of the sand-table testbed

备控制方面的能力:

(1) 温度调节任务。当用户提出“温度过高”需求时,若系统检测到窗户处于开启状态,会先执行“关闭窗户”,随后再执行“开启空调”(如图 12(a)和图 12(b)所示)。实验验证了系统在依赖关系处理上的正确性,能够保证操作顺序符合环境约束。

(2) 光照调节任务。当用户提出“室内过暗”需求时,系统能够结合时间上下文,在白天选择“打开窗帘”而在夜晚则选择“开启灯带”(如图 13 所示)。这一结果体现了系统对时序条件的感知与基于上下文的推理能力。

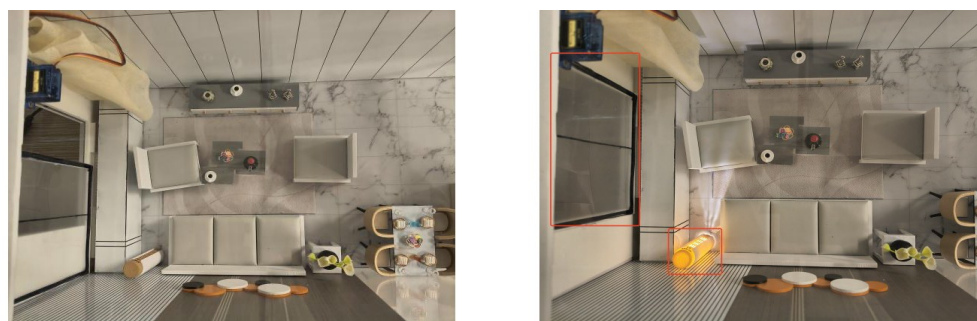
(3) 空气质量调节任务。当传感器检测到空气质量下降时,系统会自动开启新风机,并关闭厨房油烟机,以保证空气循环。该实验表明,系统能够根据环境动态变化进行多设备联动,实现对空气质量的自动调节。

(4) 环境舒适调节任务。当用户提出模糊化需求(如“让房间更舒服一些”)时,系统能够综合评估温度、湿度和光照等多维环境参数,并生成合理的操作序列。在具体执行过程中,系统先关闭窗户并确认状态,再开启空调以满足温度调节需求;同时,系统还能根据光照条件选择开启灯带,两类操作在保证依赖关系正确性的前提下实现了并行推进。结果表明,系统不仅能够解析模糊化的自然语言需求,还能在多设备协作场景中保持高效与稳定的执行表现。

3.5.3 实验结果与分析

实验结果表明,该系统能够在不同场景下根据实时环境状态与用户需求生成合理的操作序列,并稳定完成预期任务。

在温度调节任务中,系统自动生成的操作顺序与仿真测试结果保持一致,有效验证了其在设备依赖关



(a) 窗户开启状态下用户提出降温请求 (b) EnvAct关闭窗户并打开空调
(a) The user issues a cooling request while the window is open (b) EnvAct closes the window and turns on the air conditioner

图 12 温度调节任务中的依赖补全执行结果
Figure 12 Dependency completion execution results in the temperature regulation task



图 13 光照调节任务中的上下文感知执行结果:白天环境下系统选择打开窗帘;夜晚环境下系统选择开启灯带

Figure 13 Context-aware execution results in the light regulation task: the system opens the curtain during the daytime and turns on the light strip at night

系处理上的正确性;在光照调节任务中,系统能够结合环境的时间属性选择合适的动作(如白天优先开窗帘、夜晚选择开灯),体现出对环境上下文的理解与适配能力;在空气质量与舒适度任务中,系统不仅能够融合多维环境参数进行综合决策,还展现出在多设备控制下的并行执行能力,从而提升了整体响应效率。

整体来看,系统在沙盘实验中的执行成功率保持在 90% 以上,任务完成时间控制在 30 s 级别,充分验证了该框架在真实 IoT 环境下的可行性与鲁棒性。

4 结束语

本文提出了一种融合实时环境表征与大语言模型(LLM)的物联网自动化执行框架 EnvAct,通过将结构化环境建模与语义推理深度耦合,实现了物理环境与语言模型推理过程的统一。该框架由环境状态引擎、设备管理器与推理引擎组成,支持环境约束与依赖关系的动态建模,并通过合规化操作生成与反馈优化机制,提升了系统在复杂动态场景下的可靠性与适应性。

在任务集上,EnvAct 在不同大模型下的任务成功

率都达到 88%,较 Sasha 提高 40 个百分点以上,较 SAGE 提高 13 个百分点以上。在面对通信延迟与执行异常等场景的动态干扰实验中,系统能保持 81.25% 的高可靠性。在真实物理沙盘环境中,系统执行成功率保持在 90% 以上,平均操作延迟控制在 30 s 以内,进一步验证了该框架在真实 IoT 协议与设备交互条件下的可行性与鲁棒性。

总体而言,EnvAct 通过引入实时环境表征机制,有效弥补了现有 LLM Agent 在环境感知与上下文对齐方面的不足,显著提升了物联网场景下的任务规划与执行能力。尽管如此,实验结果也表明,当系统面对传感器数据漂移或设备瞬时失效等执行端点异常时,可靠性仍会受到一定影响,说明源头感知数据质量对后续语义判定与合规校正具有重要作用。因此,后续研究将面向真实大规模物联网环境开展实地部署测试,重点优化并发请求下的推理效率,并引入数据质量评估、传感器异常检测与多源状态校验机制,以增强系统在源头数据污染和设备异常条件下的韧性。同时,结合持续学习技术进一步优化环境表征机制,提升系统的泛化性与实用价值。

参考文献

- [1] Chataut R, Phoummalayvane A, Akl R. Unleashing the power of IoT: A comprehensive review of IoT applications and future prospects in healthcare, agriculture, smart homes, smart cities, and industry 4.0[J]. *Sensors*, 2023, 23(16): 7194.
- [2] Ur B, McManus E, Pak Yong Ho M, et al. Practical trigger-action programming in the smart home[C]//*Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York: ACM, 2014: 803-812.
- [3] Wang Qi, Datta P, Yang Wei, et al. Charting the attack surface of trigger-action IoT platforms[C]//*Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. New York: ACM, 2019: 1439-1453.
- [4] Huang J, Cakmak M. Supporting mental model accuracy in trigger-action programming[C]//*Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. New York: ACM, 2015: 215-225.
- [5] Brown T B, Mann B, Ryder N, et al. Language models are few-shot learners[C]//*Proceedings of the 34th International Conference on Neural Information Processing Systems*. New York: ACM, 2020: 1877-1901.
- [6] Cui Hongwei, Du Yuyang, Yang Qun, et al. LLMind: Orchestrating AI and IoT with LLM for complex task execution[J]. *IEEE Communications Magazine*, 2025, 63(4): 214-220.
- [7] K k İ, Demirci O,  zdemir S. When IoT meet LLMs: Applications and challenges[C]//*2024 IEEE International Conference on Big Data*. Piscataway: IEEE, 2024: 7075-7084.
- [8] An Tuo, Zhou Yunjiao, Zou Han, et al. IoT-LLM: A framework for enhancing large language model reasoning from real-world sensor data[J]. *Patterns*, 2026, 7(1): 101429. DOI:10.1016/j.patter.2025.101429.
- [9] Rivkin D, Hogan F, Feriani A, et al. SAGE: Smart home agent with grounded execution[PP/OL]. V2. arXiv (2024-01-19)[2026-04-27]. <https://doi.org/10.48550/arXiv.2311.00772>.
- [10] Cheng Ye, Xu Minghui, Zhang Yue, et al. AutoIoT: Automated IoT platform using large language models[J]. *IEEE Internet of Things Journal*, 2025, 12(10): 13644-13656. DOI:10.1109/JIOT.2024.3523907.
- [11] Liu Kaiwei, Yang Bufang, Xu Lilin, et al. TaskSense: A translation-like approach for tasking heterogeneous sensor systems with LLMs[C]//*Proceedings of the 23rd ACM Conference on Embedded Networked Sensor Systems*. New York: ACM, 2025: 213-225.
- [12] Ji Zhenlan, Wu Daoyuan, Ma Pingchuan, et al. Testing and understanding erroneous planning in LLM agents through synthesized user inputs[PP/OL]. V1. arXiv (2024-04-27)[2026-04-27]. <https://doi.org/10.48550/arXiv.2404.17833>.
- [13] King E, Yu Haoxiang, Lee S, et al. Sasha: Creative goal-oriented reasoning in smart homes with large language models[J]. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2024, 8(1): 1-38.
- [14] Soares D, Dias J P, Restivo A, et al. Programming IoT-spaces: A user-survey on home automation rules[C]//*Computational Science - ICCS 2021*. Cham: Springer International Publishing, 2021: 512-525.
- [15] Mi Xianghang, Qian Feng, Zhang Ying, et al. An empirical characterization of IFTTT: Ecosystem, usage, and performance[C]//*Proceedings of the 2017 Internet Measurement Conference*. New York: ACM, 2017: 398-404.
- [16] Aavild A P, Rosenkrantz de Lasson A, Moesgaard Andersen C, et al. Distributed home automation with home assistant[C]//*Proceedings of the 14th International Conference on the Internet of Things*. New York: ACM, 2025: 206-212.
- [17] Yu Haoxiang, Hua Jie, Julien C. Analysis of IFTTT recipes to study how humans use Internet-of-things (IoT) devices[C]//*Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. New York: ACM, 2021: 537-541.
- [18] Cabitza F, Fogli D, Lanzilotti R, et al. End-user development in ambient intelligence: A user study[C]//*Proceedings of the 11th Biannual Conference of the Italian SIGCHI Chapter*. New York: ACM, 2015: 146-153.
- [19] Hsu K H, Chiang Y H, Hsiao H C. SafeChain: Securing trigger-action programming from attack chains[J]. *IEEE Transactions on Information Forensics and Security*, 2019, 14(10): 2607-2622.
- [20] Minder B, Wolf P, Baldauf M, et al. Voice assistants in private households: A conceptual framework for future research in an interdisciplinary field[J]. *Humanities and Social Sciences Communications*, 2023, 10: 173.
- [21] Tamanna I, Lima S. An IoT based smart home automation using Google assistant[J]. *International Journal of Research and Innovation in Applied Science*, 2023, VIII: 112-119.
- [22] Bogdan R, Tatu A, Crisan-Vida M M, et al. A practical

- experience on the Amazon alexa integration in smart offices[J]. *Sensors*, 2021, 21(3): 734.
- [23] Celebre A M D, Dubouzet A Z D, Medina I B A, et al. Home automation using raspberry Pi through Siri enabled mobile devices[C]//2015 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management. Piscataway: IEEE, 2015: 1-6.
- [24] Venkatraman S, Overmars A, Thong M. Smart home automation: Use cases of a secure and integrated voice-control system[J]. *Systems*, 2021, 9(4): 77.
- [25] Hwang A, Oza N, Callison-Burch C, et al. Rewriting the script: Adapting text instructions for voice interaction[C]//Proceedings of the 2023 ACM Designing Interactive Systems Conference. New York: ACM, 2023: 2233-2248.
- [26] Khan I U, Afzal S, Lee J W. Human activity recognition via hybrid deep learning based model[J]. *Sensors*, 2022, 22(1): 323.
- [27] Gao Yi, Xiao Kaijie, Li Fu, et al. ChatIoT: Zero-code generation of trigger-action based IoT programs[J]. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2024, 8(3): 1-29.
- [28] Wu Qinyue, Shen Beijun, Chen Yuting. Learning to recommend trigger-action rules for end-user development[C]//Reuse in Emerging Software Engineering Practices. Cham: Springer International Publishing, 2020: 190-207.
- [29] Yusuf I N B, Jiang Lingxiao, Lo D. Accurate generation of trigger-action programs with domain-adapted sequence-to-sequence learning[C]//Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension. New York: ACM, 2022: 99-110.
- [30] Felfernig A, Polat-Erdeniz S, Uran C, et al. An overview of recommender systems in the Internet of Things[J]. *Journal of Intelligent Information Systems*, 2019, 52(2): 285-309.
- [31] Yao Yuhang, Kamani M M, Cheng Zhongwei, et al. FedRule: Federated rule recommendation system with graph neural networks[C]//Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation. New York: ACM, 2023: 197-208.
- [32] Wang Beidou, Guo Xin, Ester M, et al. Device-aware rule recommendation for the Internet of Things[C]//Proceedings of the 27th ACM International Conference on Information and Knowledge Management. New York: ACM, 2018: 2037-2045.
- [33] Rivkin D, Hogan F, Feriani A, et al. AIoT smart home via autonomous LLM agents[J]. *IEEE Internet of Things Journal*, 2025, 12(3): 2458-2472. DOI: 10.1109/JIOT.2024.3471904.
- [34] Song K, Jayarajan A, Ding Yaoyao, et al. Aegis: Taxonomy and optimizations for overcoming agent-environment failures in LLM agents[PP/OL]. V1. arXiv (2025-08-27)[2026-04-27]. <https://doi.org/10.48550/arXiv.2508.19504>.
- [35] Aghzal M, Plaku E, Stein G J, et al. A survey on large language models for automated planning[PP/OL]. V1. arXiv (2025-02-18)[2026-04-27]. <https://doi.org/10.48550/arXiv.2502.12435>.
- [36] Wei J, Wang Xuezhi, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models[C]//Proceedings of the 36th International Conference on Neural Information Processing Systems. New York: ACM, 2022: 24824-24837.
- [37] Yao Shunyu, Zhao J, Yu Dian, et al. ReAct: Synergizing reasoning and acting in language models[C/OL]//International Conference on Learning Representations (ICLR). National Science Foundation, 2023. https://openreview.net/forum?id=WE_vluYUL-X.
- [38] Aghzal M, Plaku E, Yao Ziyu. Can large language models be good path planners? A benchmark and investigation on spatial-temporal reasoning[PP/OL]. V3. arXiv (2025-02-24)[2026-04-27]. <https://doi.org/10.48550/arXiv.2310.03249>.
- [39] Jiao Kechen, Fang Zhirui, Liu Jiahao, et al. Interactive-COT: Aligning dynamic chain-of-thought planning for embodied decision-making[PP/OL]. OpenReview (2024-09-24)[2026-04-27]. <https://openreview.net/forum?id=Y4iaDU4yMi>.
- [40] Singhal K, Azizi S, Tu Tao, et al. Large language models encode clinical knowledge[J]. *Nature*, 2023, 620(7972): 172-180.
- [41] Wang Xinglin, Feng Shaoxiong, Li Yiwei, et al. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning[PP/OL]. V3. arXiv (2025-02-12)[2026-04-27]. <https://doi.org/10.48550/arXiv.2408.13457>.

作者简介



胡明明 男,1997年3月出生于安徽省明光市。现为复旦大学计算与智能创新学院博士研究生。主要研究方向为人机物融合智能。

E-mail: mmhu25@m.fudan.edu.cn



孙丙坤 男,1995年12月出生于陕西省西安市。现为复旦大学计算与智能创新学院博士研究生。主要研究方向为agent系统与大语言模型后训练算法。

E-mail: sunbingkun@m.fudan.edu.cn



滕储华 男,2003年3月出生于江西省丰城市。现为复旦大学计算与智能创新学院硕士研究生。主要研究方向为人机物融合智能。

E-mail: chteng25@m.fudan.edu.cn



沈立炜 男,1982年7月出生于上海市。现为复旦大学计算与智能创新学院副教授。主要研究方向为人机物融合系统软件、泛在计算与泛在操作系统等。

E-mail: shenliwei@fudan.edu.cn



李明辉 男,2002年11月出生于山东省临沂市。现为复旦大学计算与智能创新学院硕士研究生。主要研究方向为人机物融合智能。

E-mail: limh2002@163.com