利用大语言模型实现高效且个性化的智能家居自动化

Chaerin Yu, C hihun Choi、Sunjae Lee、Hyosu Kim、Steven Y. Ko、Young-Bae Ko 和 Sangeun O h

arXiv:2601.04680v1 [cs.HC] 8 Jan 2026

摘要—智能家居设备的普及增加了对其进行控制和管理的复杂性，从而导致用户疲劳。在此背景下，大语言模型（LLMs）能够为物联网（IoT）控制提供自然语言界面，因此成为一种颇具前景的解决方案。然而，现有基于大语言模型的方法受到大语言模型非确定性、高推理延迟与成本以及个性化程度有限等因素的影响，导致设备控制不可靠且效率低下。为应对这些挑战，我们提出了 IoTGPT，一种基于大语言模型的智能家居代理，旨在以可靠、高效且个性化的方式执行物联网命令。受人类管理复杂任务方式的启发，IoTGPT会将用户指令分解为子任务并对其进行记忆。通过重复利用已学习的子任务，后续指令能够以更少的大语言模型调用得到更高效的处理，从而提高可靠性并降低延迟和成本。IoTGPT 还支持细粒度的个性化，可根据用户偏好调整各个子任务。评估结果表明，IoTGPT 在准确率、延迟/成本和个性化方面均优于基线方法，同时减轻了用户负担。

I. 引言

OVER 在过去十年间，物联网技术的发展将各种设备——从照明和供暖到安防系统和家用电器——引入智能家居，使这些环境日益复杂。截至2025年，美国每户家庭平均拥有21台联网物联网设备 [1], ，凸显了这种复杂性的规模。随着互联设备数量不断增加，用户在有效控制和管理这些设备时面临着更高的认知负荷 [2]。因此，人们对高效且直观的物联网设备控制方法的需求日益增长，以帮助减轻用户的疲劳。

毫不意外，人们已经探索了各种实现智能家居自动化的方法。现有的物联网平台，例如 Samsung SmartThings [3], Google Home [4], 和 Apple HomeKit [5] ，提供了无需编写源代码即可定义触发-动作规则的可视化界面。虽然这类界面直观易用，适合处理简单任务，但随着所需自动化的复杂程度不断提高，它们会带来越来越大的认知负荷。另一种方法是使用语音助手，例如 Google Assistant [6] 和 Apple Siri [7], ，将用户的口头指令转换为物联网命令。这些系统基于预定义的指令模板，在解读明确且结构良好的请求（例如：“在下午 *6* 点打开客厅的灯光”）时表现良好。然而，对于更加自然但含义模糊的指令（例如：“日落后让客厅明亮起来”），它们往往难以应对，因为这需要推理用户意图，从而导致自动化执行错误或不完整。

C. Yu 和 Y. Ko 任职于韩国水原 Ajou University 人工智能融合网络系，邮编 16499（电子邮箱：dbcofls6961@ajou.ac.kr；youngko@ajou.ac.kr）。S. Lee 任职于韩国水原成均馆大学计算机科学与工程系，邮编 16419（电子邮箱：sun-jae.lee@skku.edu）。H. Kim 任职于韩国首尔 Chung-Ang University 计算机科学与工程学院，邮编 06974（电子邮箱：hskimhello@cau.ac.kr）。S. Y. Ko 任职于加拿大不列颠哥伦比亚省本拿比市西门菲莎大学计算科学学院，邮编 V5A 1S6（电子邮箱：steveyko@sfu.ca）。C. Choi 和 S. Oh 任职于韩国首尔高丽大学计算机科学与工程系，邮编 02841（电子邮箱：clgns0102@korea.ac.kr；sangeunoh@korea.ac.kr）。Sangeun Oh 和 Young-Bae Ko 为共同通讯作者。

大语言模型（LLM）正成为智能家居自动化领域的变革性力量。与传统方法不同，基于大语言模型的智能体 [8]–[12] 能够灵活地理解用户话语并推断其潜在意图，即使指令含糊不清，也能据此生成物联网命令。然而，它们也面临着一些关键局限。首先，大语言模型的非确定性会削弱可靠性：幻觉命令要求用户反复检查并纠正结果。其次，大语言模型推理的计算成本高昂，提示处理和推理都需要较长时间。现有智能体通过多步推理和重新规划，反复调用大语言模型，进一步推高了成本；在我们的实验中，一种方法处理一条简单指令最多耗时 151 秒。最后，大语言模型缺乏针对特定用户的上下文，因此难以实现个性化自动化。例如，对于“让卧室做好入睡准备”这类未明确说明细节的指令，用户对具体要求往往有不同偏好（例如调暗灯光、降低温度或减少背景噪声），而当前系统难以可靠地推断这些偏好。

为克服这些局限，我们提出了IoTGPT，这是一种基于大语言模型的智能家居代理，能够以可靠、高效且个性化的方式将自然语言指令转换为可执行物联网命令。其核心思想是任务分解与记忆。受人类将复杂任务拆分为更小、更易管理的子任务并加以学习以备将来使用的方式启发[13], [14], IoTGPT 会将指令分解为明确的子任务（例如，“让卧室做好入睡准备” → {“调节空调温度”、“调暗睡眠灯”}），这些子任务可以轻松转换为应用程序接口级别的命令。随后，这些子任务及其转换后的设备命令会一同存储在记忆中，并在今后的指令中出现类似设备控制子任务时重复使用。

这种分解带来了两大优势：*i)* 子任务可以被记忆并在不同指令间重复使用，从而减少重复的大语言模型推理，进而降低命令生成中的不可靠性、延迟和成本；*ii)* 通过子任务暴露的可配置参数，使低级设备命令能够根据给定的上下文和用户偏好进行微调。此外，在一个子任务中学到的偏好也可以指导其他子任务。例如，如果 IoTGPT 学会了用户在控制

空调时偏好“凉爽”对应 22°C，那么在使用风扇等其他制冷设备调节室温时，它也可以自动应用这一偏好，即使设备配置发生变化，也能实现自适应个性化。

为实现这一目标，IoTGPT 提出了三项关键特性：*i)* 可靠的任务分解：由于具有随机性，大语言模型可能会生成不一致的子任务或无效的设备命令。为缓解这一问题，IoTGPT 将结构化提示与纠正机制相结合，以确保命令有效。*ii)* 高效的子任务复用：为了促进子任务的高效复用，IoTGPT 将任务知识组织成抽象的层次化数据结构（任务记忆）。任务记忆以通用形式紧凑地存储子任务和设备命令，从而支持在不同任务间灵活复用。*iii)* 自适应个性化：为在设备可用性不同的情况下实现个性化，IoTGPT 将用户偏好从特定设备的设置抽象为与设备无关的环境属性。这使偏好能够跨设备迁移；例如，如果根据用户使用空调的情况学到了其对凉爽环境的偏好，那么当只有风扇可用时，IoTGPT 便可通过调节风扇转速来近似满足这一偏好。

贡献。 总而言之，我们的贡献如下：

- 我们提出了 IoTGPT，这是一种基于大语言模型的智能家居代理，通过任务分解和记忆实现可靠、高效且个性化的物联网控制。
- 我们引入了一种分层任务记忆，支持高效复用子任务，减少对大语言模型的依赖，并提升自动化的可靠性和效率。
- 我们设计了自适应个性化机制，将用户偏好抽象为与设备无关的环境属性，从而支持在异构设备配置之间迁移。
- 我们在三星 SmartThings 平台上使用 IoTGPT原型开展了实验。结果表明，与最先进的基线方法相比，任务成功率最高提升 85.43%，延迟降低 78.40%，成本降低 44%。
- 我们对 15 位参与者开展了深入的用户研究，以评估 IoTGPT。结果显示，与现有方法相比，该系统降低了认知负荷，提升了可用性，并实现了更有效的个性化。

II. 相关工作

IoTGPT 是一种用于智能家居自动化的基于大语言模型的智能体。鉴于其研究范围，我们从三个方面回顾相关工作：传统智能家居自动化、基于大语言模型的智能家居代理和基于大语言模型的任务自动化。

传统智能家居自动化。 智能家居自动化的需求日益增长，推动了各种自动化技术的发展 [15]。传统平台（如 SmartThings [3], 、Google Home [4],、Apple HomeKit [5]）要求用户通过可视化界面手动创建触发条件—动作规则。虽然这些规则简单易用，但灵活性有限，而且随着任务复杂度增加，会带来较高的认知负荷。为减轻这一负担，基于众包的解决方案 [16], [17]让用户共享规则，但由于质量控制和标准化方面的挑战，其可靠性仍存在问题。机器学习方法 [18]试图通过从传感器和设备日志中推断用户日常行为，实现个性化控制，但往往无法准确捕捉用户意图。自然语言接口 [19]–[21] 也已集成到 Siri [7] 和 Google Assistant [6], 等助手中，让用户能够通过口头指令直观地进行控制。然而，这些接口依赖预定义的指令模板，难以适应非正式表达，而此类表达往往含糊不清，因此降低了可用性。因此，传统方法仍然僵化且繁琐，难以处理复杂或含糊的请求。这凸显了开发能够自主处理意图理解与适应的系统的必要性，从而尽可能减少对用户手动干预的依赖。

基于大语言模型的智能家居代理。 近期研究探索了利用大语言模型实现智能家居自动化 [8]–[12]。一些研究 [10], [11] 利用大语言模型的推理能力，将各种自然语言请求转化为自动化规则；另一些研究 [12]则利用大语言模型提供可视化选项，以实现指令消歧。然而，这些方法都需要用户大量参与，要求用户验证生成的规则，或手动选择预期的解释。Sasha [8] 通过用户请求澄清和相关设备筛选等多步推理，减轻了这一负担。然而，它仍未考虑用户上下文和个性化。此外，其单体式方法将整条指令放在单个推理轮次中处理，因此容易出错，尤其是在处理复杂任务时。为解决这一问题，SAGE [9]会动态规划推理模块序列，以灵活处理任务，并利用过往设备交互实现个性化。然而，这种方法严重依赖大语言模型，既带来计算开销，也因其非确定性而导致错误累积。此外，其设备特定个性化在设备配置发生变化时适应性有限。相比之下，IoTGPT将指令拆解为可复用的子任务并缓存起来，以供将来使用，从而尽量减少重复的大语言模型推理。此外，不同于以往仅限于设备特定历史记录的研究，IoTGPT将用户偏好抽象为设备无关属性，从而能够在异构设备之间灵活实现个性化。

基于大语言模型的任务自动化。 大语言模型已被用于探索在不同计算领域实现端到端任务自动化。例如，移动智能体 [22]–[26] 利用大语言模型解析图形用户界面组件，从而自动控制移动应用。另一类研究 [27]–[30] 通过分析网页组件（如 HTML DOM 树结构），专注于网页浏览器中的任务自动化。同样，大语言模型通过监控用户屏幕 [31]–[34], 实现了桌面生产力软件中的操作自动化。这些研究展现了大语言模型通过屏幕解析，在软件环境中实现任务自动化的潜力。值得注意的是，与 IoTGPT类似，其中一些研究采用任务分解策略来处理复杂工作流。然而，IoTGPT 所针对的智能家居自动化面临着独特挑战：需要对动态物理环境（例如环境条件）进行推理、协调异构设备，并支持自适应个性化。IoTGPT通过将任务分解与上下文感知记忆相结合，应对这些独特的复杂问题，从而确保在物理空间中实现可靠且个性化的自动化。

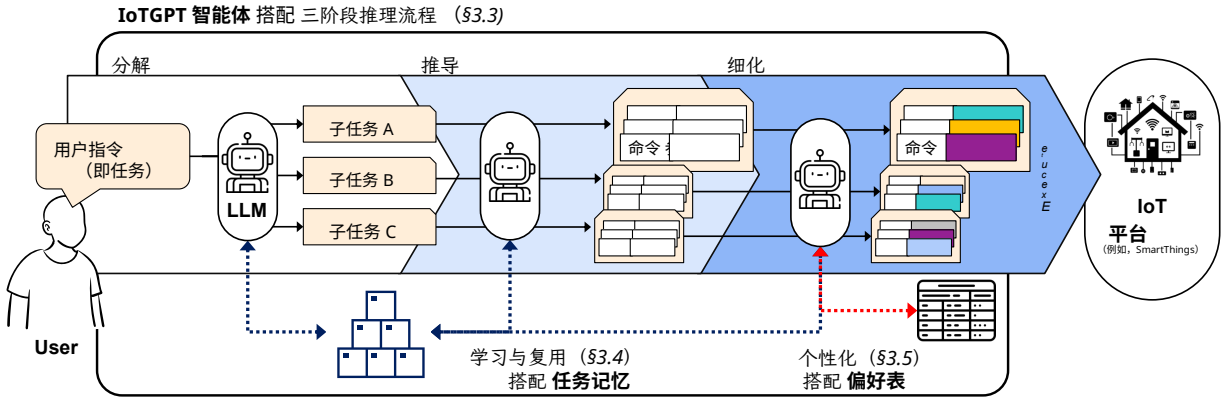


图 1. IoTGPT 的系统概览。它通过三阶段推理流程（拆解–推导–细化）将自然语言指令转换为物联网命令，并利用任务记忆实现学习与复用，利用偏好表实现个性化。

III. IOTGPT 设计

我们介绍 IoTGPT，这是一款基于大语言模型的智能家居代理，旨在实现可靠、高效且个性化的物联网控制。本节将详细介绍其设计原则与挑战。

A. 系统设计概述

设计原则。 面对复杂且含糊的任务时，人们很少会一次性解决；相反，他们会将任务分解为更小、更易处理的子任务 [13][14]。受这一认知过程启发，我们设计了 IoTGPT，使其以类似的方式处理用户指令，从而实现智能化、个性化的智能家居自动化。图 1 展示了 IoTGPT 的总体设计。其核心是一个三阶段推理流程，逐步将自然语言指令转换为可执行物联网命令，例如触发-动作规则、直接控制命令或设备查询[1]。每个阶段都使用大语言模型逐步细化中间结果，最终生成能够有效体现用户偏好的个性化输出。三个阶段如下：

- **分解**：给定一条用户指令，IoTGPT 会将其拆分为若干子任务，这些子任务是具体、面向设备的操作，共同完成该指令。如果指令含糊不清（例如“让卧室做好入睡准备”），IoTGPT会利用大语言模型的常识提出合理的子任务作为基线方案（例如，{“调节空调温度”、“设置加湿器档位”、“调暗睡眠灯”}）。这一方案是起点，而非最终决定；我们也承认，最初的猜测可能无法完全符合用户的意图。
- **推导**：对于每个子任务，IoTGPT 会生成可执行的 API 级设备命令，并为未指定的参数设置抽象占位符（例如“调节空调温度” ⇒ [打开空调 → 将模式设置为 *[模式值]*→ 将温度设置为 *[温度值]*]）。此阶段侧重于生成有效的命令结构，同时保留未指定参数的可配置性。
- **细化**：IoTGPT 会根据从上下文和过往交互中推断出的用户偏好，填入参数，以实现这些命令的个性化。不过，如果没有偏好证据（例如，首次使用），IoTGPT会采用保守的默认设置，并标记不确定的选择供用户审核。

至关重要的是，在IoTGPT 完成优化阶段后，会进入人工参与环节，用户可以审核提案，并在必要时修改子任务（例如添加“播放音乐”）或调整参数，以符合自己的偏好。这种设计将用户干预集中到一次最终审核中，从而减轻用户负担，而不必要求用户逐步监督。此外，这些反馈会同时更新系统所学习的任务知识（即任务记忆）和偏好档案，使后续指令能够更好地反映用户偏好。最终确定后，生成的命令会被发送至物联网平台（例如 Samsung SmartThings）。当触发-动作规则的条件满足时，这些规则会被安装并执行，而直接控制命令和设备查询则会立即执行。

任务分解的优势。遵循这一架构，我们的任务分解方法带来了两大优势。首先，它能够在不同任务间高效复用知识，从而提升效率和可靠性。为实现这一优势，IoTGPT利用其任务记忆记录此前学到的子任务及其关联的命令结构。例如，在处理一条指令时，IoTGPT 一旦推导出子任务“调节空调温度”对应的设备级命令，便可在另一条指令（例如“保持厨房凉爽”）中复用这些知识，同时根据上下文（例如烹饪）调整参数，而无需从头重新计算。随着任务记忆不断增长，IoTGPT 对重复进行大语言模型推理的依赖会降低，从而缓解物联网命令生成中的不可靠性和延迟问题。

其次，任务分解能够在子任务级别实现细粒度个性化。由于子任务通过低级命令暴露了可配置参数，IoTGPT可以根据用户偏好调整这些数值。此外，一旦学会的用户偏好，还可以灵活迁移到未来任务中的类似子任务。例如，如果通过子任务“调节空调温度”学会了用户对“凉爽”环境的偏好，那么之后也可以将这一偏好应用于另一个同样会影响

[1]触发-动作规则是由事件驱动的自动化（例如，“有人在客厅时打开灯光”）。直接控制命令是立即执行的设备操作（例如，“让卧室适合睡眠”）。设备查询用于询问智能家居的状态（例如，“室温是多少？”）。

温度的子任务，例如“调节风扇转速”，以便在没有空调的情况下使用。这种子任务级个性化使IoTGPT能够随着时间推移捕捉累积的偏好，并将其应用于新任务。

B. 技术挑战

尽管任务分解具有明显优势，但将这一设计付诸实践仍会带来若干技术挑战：

C1. 如何可靠地分解任务？ 生成有效物联网命令的一项关键挑战在于，大语言模型往往无法稳定地将自然语言指令分解为子任务，并推导出相应的低级设备命令。这是因为大语言模型容易产生歧义，且具有非确定性：当要求其“让卧室准备好入睡”时，可能会引入当前环境中无法执行的子任务（例如，在未安装智能百叶窗的情况下“调节窗户百叶窗”），或生成无效的设备命令（例如，为空调设置不受支持的模式）。此类行为可能导致生成的物联网命令不完整或不可行。因此，要确保可靠的任务分解，需要精心设计提示策略并采用适当的纠正机制（详见第IV节）。

C2. 如何高效支持子任务复用？ 即使子任务及其设备命令能够被正确生成，由于情境差异，在不同任务之间复用它们仍然具有挑战性。同一个子任务（例如“调节空调温度”）可能出现在不同情境中，例如“保持厨房凉爽”与“让卧室准备好入睡”。如果分别存储每个特定情境的变体，任务记忆可能会迅速膨胀，导致过高的空间开销，并使跨相似情境复用知识变得困难。另一方面，如果不考虑情境就记忆子任务，复用可能会忽略情境细节，无法恰当地体现个性化。因此，高效复用需要精心设计任务记忆，使其能够对已存储的任务知识进行情境感知的适应。这包括有选择地检索相关任务知识，并对其进行调整以适应当前情境（详见第V节）。

C3.如何自适应地实现物联网命令的个性化？ 智能家居系统中的传统个性化方法依赖设备交互历史来捕捉用户偏好，通常以单个设备为级别。然而，应用这些设备-

具体偏好存在两项局限。首先，为某一设备捕获的偏好无法轻易转移到具有不同功能的其他设备上；例如，了解用户偏好的空调设置，并不能直接说明在只有风扇可用时应如何配置风扇。其次，由于偏好取决于情境，仅依赖设备使用历史会导致难以在不同情境下应用正确的偏好。为克服这些局限，IoTGPT 将用户偏好抽象为与设备无关的环境属性（例如温度、亮度），并针对不同情境（例如睡眠和锻炼）分别维护这些偏好。这种情境感知抽象能够实现偏好在异构设备和不同情境之间的灵活迁移（详见第六节）。

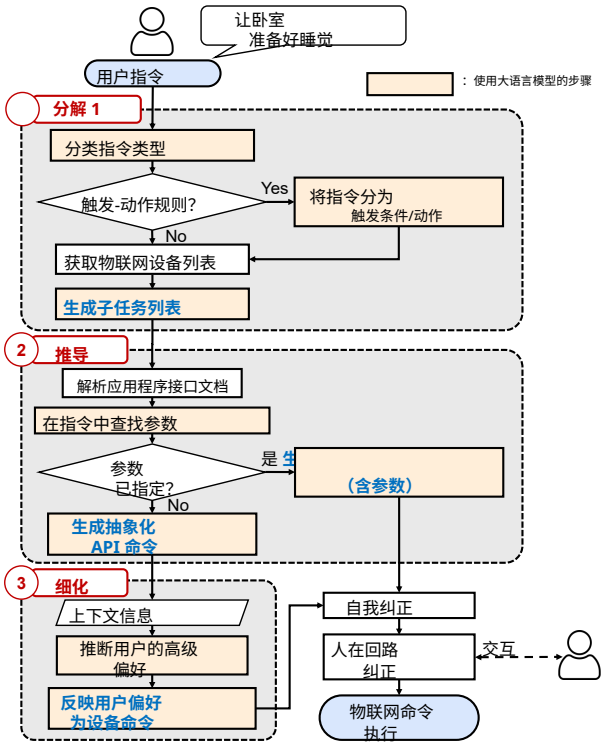


图 2. IoTGPT 流程的工作流

IV. 任务分解流水线

为了将自然语言指令转换为物联网命令，IoTGPT 采用了如图 2 所示的分解–推导–优化流程。本节概述该流程的运行方式，以及IoTGPT 如何应对大语言模型固有的不可靠性所引发的问题。

A. 流水线的三个阶段

分解阶段。 分解阶段的目标是根据给定的用户指令生成一组可执行子任务。大语言模型经过大规模通用知识训练，能够通过常识推理，以可接受的准确率推断出合理的子任务。为此，IoTGPT首先提示大语言模型将指令分类为三种类型之一：触发–动作规则、直接控制命令或设备查询。确定类型后，IoTGPT 会通过物联网平台的 API 调用，获取当前连接到环境的物联网设备列表。该列表包含设备及其可用功能和规格，并作为上下文提供给大语言模型。通过将推理过程建立在实际设备列表的基础上，IoTGPT降低了引入当前环境中无法执行的子任务的风险。在指令类型和设备信息的指导下，大语言模型随后生成完成该指令所需的子任务，以及执行这些子任务所需的设备。具体而言，对于触发–动作规则，大语言模型会先被提示将指令拆分为触发条件和动作部分，然后分别为每个部分生成相应的子任务。

例如，给定指令“让卧室做好入睡准备”，虽然其中对于应该采取哪些具体动作的说明有些模糊，大语言模型仍然可以利用

常识推断出一组合乎情理的子任务。分解阶段的输出将是采用 JSON 格式的子任务列表，例如“调节空调温度”、“设置加湿器档位”和“调暗睡眠灯”，每项都关联相应的设备，如下所示：

```
{"CommandType": "直接控制命令", "Action": { "name": "
```

推导阶段。推导阶段的目标是将上一阶段获得的每个子任务转换为一系列严格符合物联网平台 API 规范的低级设备命令。为此，IoTGPT会向大语言模型提供三项输入：*i)* 用户指令、*ii)* 子任务列表，以及*iii)*API 文档。由于 API 文档可能篇幅庞大（例如，Samsung SmartThings 的 API 文档有 9.3MB [35]），IoTGPT 采用检索增强生成（RAG）流程来解析 API 文档，并检索与每个子任务相关的内容。

生成低级命令时，为其分配合适的参数值至关重要。如果指令中明确指定了这些值，就可以直接映射到相应的参数。然而，用户指令往往含糊不清，且缺少细节。持续向用户询问缺失信息会造成过多的交互开销。为此，IoTGPT 会将值保留为抽象占位符，之后在优化阶段填入这些值，以反映用户偏好和设备上下文。

继续前面的示例，子任务“调节空调温度”可以转换为以下命令序列：[打开空调 → 将模式设置为*[模式值]* → 将温度设置为 *[温度值]*]。由于原始指令中未指定参数值，因此会将其表示为抽象占位符。

```
{"subtask": "调节空调温度", "commands":[{"desc": "打开空调", "
```

优化阶段。优化 阶段的目标是将低级命令进行个性化，使其与用户偏好保持一致。为此，IoTGPT 向大语言模型提供提示词，其中包括*i)* 子任务及其低级命令，以及*ii)*用户偏好的结构化摘要。这些偏好从过去的设备交互中提取，并以环境属性（例如温度、亮度、噪声）的形式表示。根据这些偏好，对抽象命令的参数值进行细化。例如，命令中的参数槽位（如[将温度设为 *[温度值]*] ）会填入个性化值（例如 20°C）；如果缺少特定上下文的偏好，则填入默认值。第六节将对这一 个性化过程作出更详细的说明。

B. 两步纠错流程

即使采用结构化流水线，IoTGPT仍可能生成不完美的输出。首先，由于大语言模型具有随机性，生成的命令可能表现出不一致或错误的行为（例如，API 级错误）。其次，个性化需求可能未得到充分说明：对于给定的上下文，系统可能缺乏足够的偏好证据，从而采用保守的默认设置，或选择不匹配的参数。为解决这两个问题，IoTGPT采用两步纠错流程，将自动验证与可选的用户干预相结合。

自动自我纠正。IoTGPT首先尝试利用虚拟智能家居环境进行自动自我纠正。该环境由模拟设备构成，用于仿真实际部署。在将命令应用于物理环境之前，IoT-GPT 会先在虚拟环境中执行这些命令，并检查生成的执行日志，以发现 API 违规和运行时故障。检测到错误时，IoTGPT 会将错误消息反馈给大语言模型，并提示其仅修改命令序列中有问题的部分。这种由日志驱动的修订会持续进行，直至达到重试次数上限（例如 3–5 轮），从而在无需用户参与的情况下提供轻量级保障。

人在回路纠正。如果自行纠正后仍有错误，或者用户希望覆盖含糊的个性化选择，IoTGPT 就会升级到交互式纠正步骤。大语言模型会简要说明生成的命令，让用户通过自然语言反馈进行干预。用户可以指出需要纠正之处，以解决剩余的有效性问题（例如，删除错误的子任务），也可以通过调整参数或添加新子任务，进一步个性化自动化。在后一种情况下，IoTGPT会从推导阶段重新调用整个流程，以纳入新要求。这些反馈既能纠正当前任务，也会更新系统的任务记忆和偏好档案，以便用于未来的指令。

五、分层任务记忆

三阶段流水线生成的任务知识会系统地归档到IoTGPT的任务记忆中。这些记忆充当知识库，使IoTGPT 能够通过复用先前学习的子任务和设备命令，让后续的物联网命令生成更加可靠、高效。本节将介绍IoTGPT 如何组织和存储流水线结果，以及如何调用这些知识来生成未来的物联网命令。

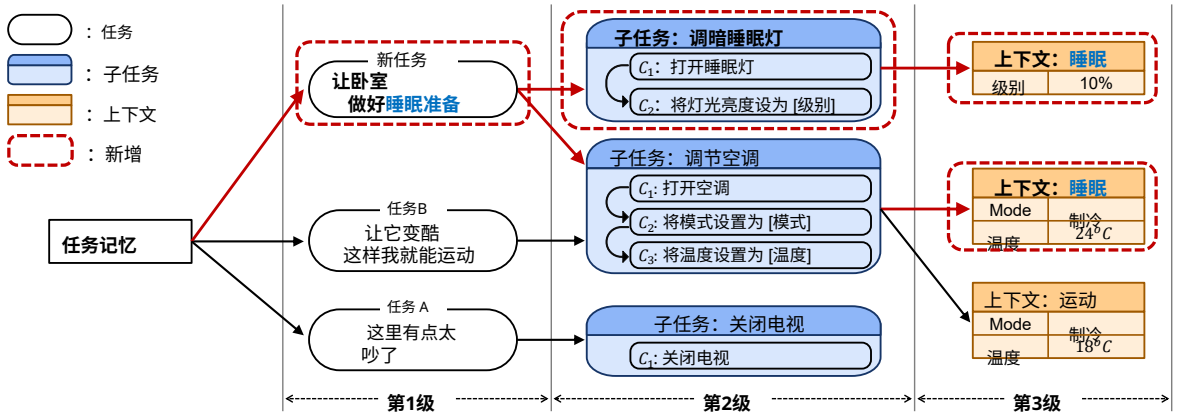


图 3. IoTGPT 的任务记忆结构

A. 任务记忆结构

如图 3 所示，IoTGPT将其任务记忆组织为一个包含三个深度级别的分层有向无环图（DAG）：任务 （第 1 级）、子任务 （第 2 级）和上下文 （第 3 级）。每个级别以不同粒度保存知识，从而实现复用和个性化。

任务级知识。每个任务节点都对应一条以自然语言发出的用户指令。当收到新指令时，IoTGPT 会先通过基于文本嵌入的余弦相似度，将收到的指令与已存储的任务进行比较，以检查记忆中是否已存在类似任务，然后再执行分解阶段。如果存在类似任务，则可以重用已存储的知识（如后文的召回流程所述）；否则，系统会创建一个新的任务节点并执行分解阶段。

子任务级知识。子任务节点在分解阶段创建，表示从任务中提取的子任务操作。子任务节点与其所属的任务节点相连。重要的是，子任务并不局限于单个任务，而是可以由多个任务共享。例如，当分解阶段生成一个新的子任务时，IoTGPT 会检查记忆中是否存在相似的子任务节点；如果找到匹配项，则只需在当前任务节点与匹配的子任务节点之间添加一条新边。这样，先前存储的子任务便可以在不同任务间直接重用。如果没有找到匹配项，则会创建一个新的子任务节点。

每个子任务节点都存储推导阶段生成的低级设备命令序列。当指令中明确指定参数值，且这些参数值以具体值的形式出现在生成的命令中时，IoTGPT 会在将命令存储到任务记忆之前，用抽象占位符（例如 [温度值]）替换这些具体值。这种泛化保留了命令序列的结构，同时允许稍后填充参数槽位。

上下文级知识：在优化阶段添加上下文节点，以捕获命令参数在不同指令情境（即指令的目标或意图）下的个性化方式。例如，在处理指令“让卧室准备好进入睡眠状态”时，IoTGPT使用大语言模型提取代表指令情境的关键词“睡眠”，并创建相应节点。该节点存储个性化参数值（例如，[模式值] = “制冷”、 [温度值] = “$20°C$”）。如果已存在具有相同关键词的节点，IoTGPT 可以直接复用其中存储的值。

此外，人在回路纠正带来的更新也会反映在任务记忆中。当用户添加/删除子任务或微调参数时，相应的子任务节点或上下文节点也会随之更新。因此，纠正不仅会应用于当前执行，还会为未来重用而不断积累，使任务记忆最终保留经用户验证的知识。

B. 任务知识回忆

在回忆任务知识时，IoTGPT 必须考虑环境或用户上下文可能不同于最初记忆该知识时的情况。例如，某些设备可能因故障而不再可用，或者同一子任务可能出现在不同的指令情境下。因此，根据具体情况确定知识应复用到哪个深度级别非常重要。当收到新指令时，IoTGPT 按以下方式进行：

任务/子任务级复用。如果内存中存在匹配的任务，或获取的某些子任务与已存储的子任务相匹配，IoTGPT 就会复用相应的子任务知识。在这种情况下，可以跳过该任务的分解阶段，或跳过这些子任务的推导阶段。但是，如果与子任务相关的某些设备不可用（例如发生故障），则会排除这些子任务。IoTGPT 随后会通知用户，并通过大语言模型建议替代子任务。

上下文级复用：对于可复用的子任务，IoTGPT 会通过比较上下文关键词，检查是否也能召回相关的上下文知识。如果存在匹配的上下文节点，系统会将子任务节点中抽象出的低级命令与上下文节点中的参数值相结合，从而避免重复进行个性化设置。否则，IoTGPT 仅复用深度为第2级的子任务级知识，并保留抽象的参数槽位。随后，这些槽位会在优化阶段通过个性化进行填充。

借助这种记忆回忆机制，IoTGPT 减少了对重复大语言模型推理的依赖，从而实现了更加稳定、高效的物联网命令生成。

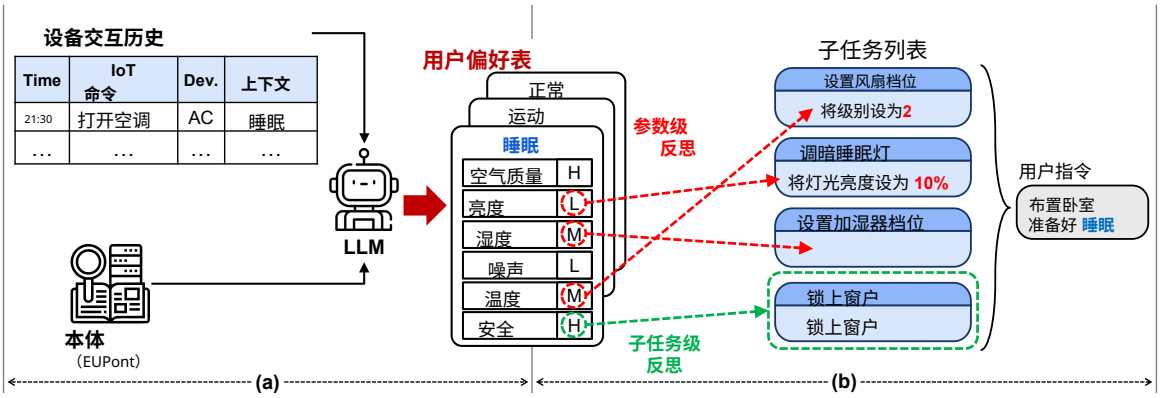


图 4. IoTGPT 的自适应个性化

VI. 自适应个性化

个性化对于智能家居自动化不可或缺，因为即使是相同的子任务，也可能因用户偏好和上下文不同而需要不同的参数值。然而，传统方法通常依赖设备特定交互历史，因此若缺少先前的使用数据，就难以将个性化扩展到其他设备。为克服这一局限，IoTGPT 将用户偏好抽象为与设备无关的环境属性（例如温度、亮度）。这种高层表示能够灵活地将偏好迁移到异构设备上。本节将介绍IoTGPT 如何从环境属性的角度提取用户偏好，并在优化阶段将其应用于物联网命令的个性化处理。

A. 用户偏好提取

如图 4(a) 所示，IoTGPT 通过向大语言模型提供*i)* 设备交互历史，以及*ii)* 物联网命令与环境属性之间的关系，以提取高层表示的用户偏好。为捕捉这些关系，我们利用 EUPont 本体 [36], ，该本体定义了影响智能家居环境的六种环境属性：空气质量、亮度、湿度、噪声、温度，以及安全。具体而言，该本体指出了每条物联网命令会影响哪些属性，以及这些属性会增加还是减少。例如，调节空调温度的主要效果是降低温度。通过将本体知识作为指导提供，IoTGPT 使大语言模型能够推断设备历史记录中物联网命令的影响。然而，提取偏好也带来了若干问题：

依赖情境的偏好。用户偏好通常会随情境而变化。例如，用户睡眠时偏好的温度可能与运动时的偏好有很大差异。为捕捉这些变化，我们的设备交互历史不仅包含原始设备命令，还包含指令情境（即指令的目标或意图）。这样，大语言模型就能提取不同情境下的偏好，例如睡眠、运动或正常。此处，正常表示不区分情境的总体偏好。

定义属性等级。 从过去的物联网命令中推断出的偏好，必须表示为每种环境属性的属性等级。一种直接的方法是使用简单的基于规则的启发式方法，将设备参数范围划分为离散等级（例如，低、中等、高）。然而，由于异构设备提供了多种多样的参数范围（例如，以 °C 表示的温度、以 % 表示的湿度、以 lux 表示的亮度），且不同属性的离散等级边界可能各不相同，因此定义适当的划分规则颇具挑战。此外，对于安全等属性，其等级并非由参数值决定，而是由更广泛的使用模式决定，例如锁闭窗户的频率或家庭摄像头的使用时长。手工制定的规则无法捕捉这种多样性和上下文依赖性。为此，IoTGPT利用大语言模型进行少样本提示，提供属性—等级映射的示例（例如温度、湿度和安全）。这使大语言模型能够灵活地划分参数范围，或将使用模式解读为适当的属性等级。

通过这一过程，IoTGPT会构建表格，将环境属性映射到偏好的级别（例如，温度 → 低）。这些表格会针对不同情境（例如，睡眠、锻炼）分别维护，使IoTGPT 能够捕捉用户偏好在不同情境下的细微差异。为使这些映射保持最新，提取过程会定期运行（例如，每天一次），或在偏好过时后的优化阶段按需运行。

B. 用户偏好反映

如图4(b)所示，在优化阶段，IoTGPT会向大语言模型提供三个输入：*i)* 指令、*ii)*子任务，以及*iii)* 根据上下文选取的偏好表。为选择偏好表，IoTGPT 会检查指令中的上下文关键词是否与偏好表中定义的任一关键词匹配。如果找到匹配项，则使用相应的偏好表；否则，系统默认使用正常 上下文偏好。随后，IoTGPT 会通过两种方式反思偏好：**参数级反思**。 在此级别，大语言模型会参考选定的偏好表，调整给定子任务的参数值。例如，在只有风扇而没有空调的睡眠环境场景中，即使没有风扇的使用历史，IoTGPT仍可调整风扇转速，使其符合表中指定的偏好温度水平。如第V节所述，个性化参数值随后会存储在任务记忆的上下文节点中。

子任务级反思。 除了调整参数外，IoT-GPT 还会根据用户偏好引入额外的子任务。例如，如果用户特别重视安全属性，大语言模型可能会添加一个新子任务，例如“锁好窗户”，而该任务在分解阶段并未考虑。随后，这个新子任务会进入推导阶段以生成相应命令，并在必要时进行参数级反思。这样，偏好反思通过加入个性化调整，弥补了分解阶段可能忽略的内容。

值得注意的是，当没有与指令匹配的上下文特定偏好表时，IoTGPT 会以正常偏好作为基线。不过，随着时间推移，用户可以通过IoTGPT的人在回路纠正，在不同情境下微调参数值。这些调整会生成更多交互日志，并将其输入偏好提取过程。该过程随后会更新现有表格或创建新的上下文特定偏好表，使其逐渐偏离正常情境。

VII. 实现说明

我们实现了IoTGPT 的一个原型，由一款通过 Wi-Fi 连接到后端服务器的 Android 移动应用构成。移动应用作为面向用户的界面：接收自然语言指令并将其转发至后端。后端负责管理任务记忆，并执行三阶段推理流程（分解–推导–优化）来处理每条指令。在推理方面，我们为IoTGPT设计了一个与骨干大语言模型交互的模块化接口。默认情况下，我们的原型使用 GPT-4o [37], ，但无需更改系统其余部分，即可将骨干模型替换为其他大语言模型。我们还将解码温度设为 0.0，以确保输出一致且确定。对于任务知识检索，我们使用 text-embedding-3-small 模型实现了基于文本嵌入的相似性搜索 [38], ，通过计算已存储节点与当前输入之间的余弦相似度，识别可复用的子任务或情境。最后，后端通过三星 SmartThings API [35], 与智能家居环境交互，同时将物联网通信层模块化，以便轻松将IoTGPT扩展至 Google Home 或 Apple HomeKit 等其他生态系统。我们的原型部署在作为客户端设备的 Google Pixel 6 智能手机上，后端服务器配备英特尔 i7-13700F CPU 和 32 GB 内存。

VIII. 性能评估

我们对IoTGPT 进行了定量评估，以考察其将自然语言指令转换为物联网命令的能力。我们从准确率、延迟和成本方面，将IoTGPT 与现有的基于大语言模型的智能家居代理进行了比较。为在丰富多样的命令空间中开展受控测试，我们使用 SmartThings Edge Driver [39], 构建了一个虚拟智能家居，模拟了 26 种不同类别的设备，包括照明、供暖和空调。

A. 对比研究

1）实验设置：在对比研究中，我们按如下方式配置了实验环境：

基线方法。 我们将IoTGPT 与两个基于大语言模型的智能家居代理进行了比较：Sasha [8] 和SAGE [9]。Sasha 是最早采用推理流程的基于大语言模型的智能家居代理之一，但它将每条指令作为单个步骤执行，不进行任务分解。SAGE是我们实验时最先进的框架，它采用动态方法，由大语言模型在运行时规划推理模块的执行顺序。尽管这种方法支持灵活分解，SAGE却不会记忆先前执行中获得的知识；相反，它会针对每条指令从头开始重新规划。我们使用了SAGE的开源版本 [40] 。由于无法获取Sasha的源代码，我们根据其原始论文中详细介绍的流程架构重新实现了该系统。

数据集。 我们使用了Sasha 和SAGE的数据集。这些数据集中的每项任务都包含一条自然语言指令，以及由人工标注者标注的真实设备命令。我们排除了与 SmartThings 平台不兼容的任务。此外，由于原始数据集中的触发-动作案例相对较少，我们额外创建了 15 项任务。因此，最终数据集共包含 97 项任务：35 项来自Sasha，47 项来自SAGE，另有 15 项由我们新增。这些任务分为三类：55 条直接控制命令、32 条触发-动作规则和 10 项设备查询。

指标。 为了严格评估准确率，我们定义了四项互补指标。任务成功率（*STR*）是所有必需的真实标注命令均被正确生成并执行（不考虑参数个性化）的任务所占比例。过度控制率（*ECR*）是包含真实标注中没有的额外命令的任务所占比例；而控制不足率（*ICR*）是至少缺少一条必需命令的任务所占比例。语法错误率（*SER*）是包含至少一条违反 SmartThings API 规格（例如存在语法错误）的命令的任务所占比例。请注意，ECR、ICR 和 SER 并非互斥指标，因为同一任务可能同时包含不必要的命令、遗漏必需命令并存在语法错误。在效率方面，我们还测量了延迟（从发出指令到虚拟智能家居中命令执行完毕的端到端时间）和成本 （根据 API 提供商的定价，按所有模型调用汇总的输入/输出词元数计算的每条指令成本，单位为美元）。

流程。我们分两个阶段评估IoTGPT：IoTGPT-Cold和IoTGPT-Warm。在IoTGPT-Cold阶段，智能体首次处理所有指令，同时构建任务记忆，并在知识可用后立即加以复用。随后，IoTGPT-Warm阶段利用已填充完整的任务记忆重新处理相同任务，但使用经过改写的数据集版本，以防止系统依赖简单的字符串匹配。具体而言，我们使用GPT-4o改写自然语言指令，同时保留其语义意图。这促使系统利用基于语义理解的知识。相比之下，Sasha 和SAGE 对每条指令只处理一次，不区分阶段。为确保比较公平，我们将其拆分——

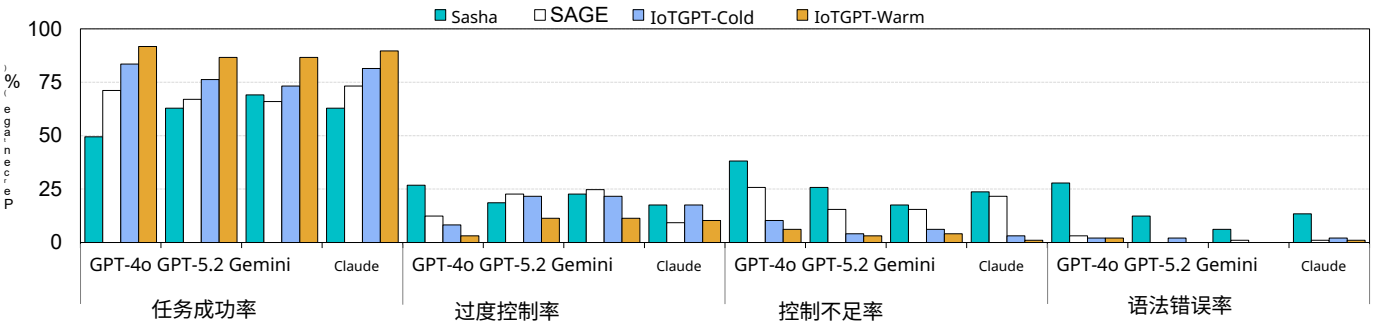


图 5. Sasha、SAGE 和 IoTGPT 的平均准确率

禁用了所有用户反馈步骤，并在每次运行中对所有方法应用相同的骨干模型。此外，为验证泛化能力，我们使用四种不同的大语言模型（GPT-4o [37], GPT-5.2 [41], Gemini 3 flash [42], 和 Claude Opus 4.5 [43]）进行了评估，并对三次运行的结果取平均值。

2）准确率：图 5 展示了三种方法在四种大语言模型骨干模型上的四项准确率指标。除 Gemini 3 flash 外，Sasha在各模型上的 STR 始终最低（例如，在 GPT-4o 上为 49.48%）。由于它以整体方式处理指令，不进行细粒度分解，因此在面对模糊或复杂任务时，Sasha 往往会遗漏必要的设备命令，并产生无效语法，导致其在智能体中具有最高的 ICR 和 SER。SAGE 借助更先进的推理模块并动态规划执行顺序，通常能比Sasha 提高 STR。然而，它依赖大语言模型，往往会生成较长的规划链，而模型的随机性会因此增加错误累积。于是，SAGE 仍表现出相对较高的 ICR，一些必要命令会被遗漏，任务也因此无法完成。

相比之下，IoTGPT-Cold 的 STR 高于Sasha 和SAGE，在四种骨干模型上的最高值达到 83.51%。它的 ICR 和 SER 也更低，表明遗漏的命令和语法错误更少。然而，IoTGPT-Cold在各模型上的 ECR 仍不可忽视，范围为 8.25% 至 21.65%。我们检查后发现，许多这类“过度”命令并非有害错误，而是可能有用的补充。例如，在处理任务“如果家里没人，就节约能源”时，IoTGPT-Cold 不仅会生成关闭电器的真实标注命令，还会额外生成一条门锁命令。尽管这条命令不在真实标注中，但有些用户可能认为它对完成任务至关重要。根据我们严格的 ECR 定义，这类输出会受到惩罚，但也凸显了IoTGPT-Cold为满足用户需求提供更丰富支持的潜力。

至关重要的是，IoTGPT 的准确率在IoTGPT-Warm 阶段进一步提升，该阶段利用任务记忆来复用先前学习的知识。具体而言，在评估的各个模型中，与IoTGPT-Cold 相比，STR最高提升了18.31%，这直接归因于对重复大语言模型推理的依赖减少。这些结果表明，随着
随着任务记忆不断增长，IoTGPT可以通过积累可复用的知识持续改进。

有趣的是，我们观察到不同系统对骨干模型表现出明显不同的敏感性。由于Sasha 依赖相对简单的

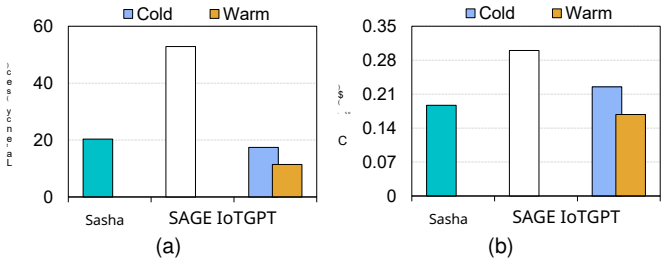


图6。(a) Sasha、SAGE 和 IoTGPT 的平均延迟。(b) Sasha、SAGE 和 IoTGPT 的平均成本。

提示设计，其绩效高度取决于骨干模型的能力。例如，在 Gemini 3 flash 下，Sasha 在 STR 方面优于SAGE ，这主要是因为增强的推理能力显著减少了语法错误。然而，尽管有所提升，Sasha的 ICR 仍然最高，表明单体推理仍容易遗漏所需的命令。相比之下，随着 GPT-5.2 和 Gemini 3 flash 等能力更强的模型的使用，IoTGPT的 STR 反而出现了违反直觉的下降。这一现象看似矛盾，实际上源于这些模型更先进的推理能力：它们往往会生成更多额外的上下文感知子任务；这些新增子任务虽然可能有用，但并未包含在固定的真实标注中，因此导致 ECR 上升并使 STR 降低。

3) 延迟与成本：图6展示了使用 GPT-4o 作为骨干模型时，各方法每条指令的平均延迟和成本。由于严重依赖大语言模型动态规划推理模块，SAGE 在这两项指标上的绩效最差。尽管这种灵活性偶尔能够提升推理质量，但也带来了大量额外开销，尤其是在规划链变得过长的复杂任务中。事实上，我们观察到有些指令耗时长达151秒，每条指令的成本最高达到4.2美元（例如“如果家里没人，就节约能源”）。相比之下，Sasha 的延迟和成本均低于SAGE，这主要得益于其简单的单体流水线，可直接处理指令而无需进行分解。然而，如准确率结果所示，这种简单性是以牺牲准确率为代价的。

与此同时，IoTGPT-Cold 呈现出一种有趣的权衡。由于多阶段流水线的令牌用量累积，Sasha 的成本低于它，但它的延迟更低。我们认为，这一反直觉的结果是因为每个阶段都使用了结构更清晰、内容更详尽的提示词；尽管包含多个阶段，这些提示词仍能缩短大语言模型的推理时间。此外

，IoTGPT -暖 通过利用任务记忆展现出显著的效率提升 -

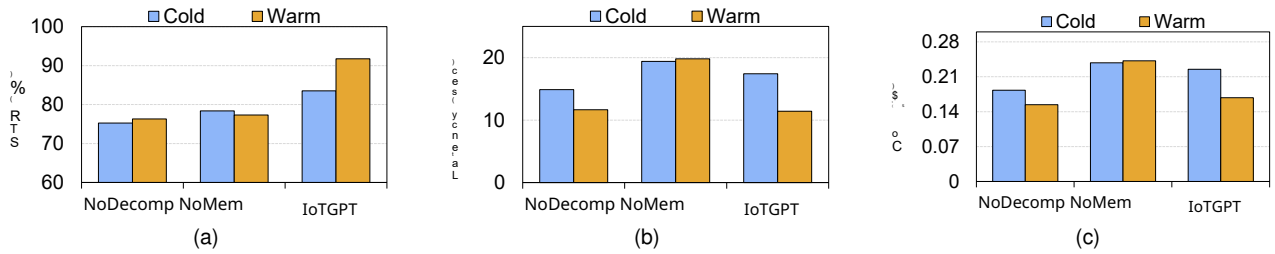


图 7. IoTGPT 在冷启动（冷）和热启动（暖）设置下的消融结果

，在所有方法中实现了最低的延迟和成本。与 IoTGPT- 冷相比，延迟和成本分别降低了 34.44% 和 25.33%。这一降低源于对已学习子任务的重复利用，无需反复进行大语言模型推理。总体而言，这些结果表明，随着任务记忆不断积累可复用的知识，IoTGPT 的效率会逐步提升。

B. 消融研究

我们进行了消融研究，以分析IoTGPT关键特性的贡献。具体而言，我们移除了任务分解和任务记忆，以评估它们对准确率、延迟和成本的影响。请注意，本消融研究未纳入个性化，因为它依赖主观的用户体验，并通过第 IX-A 节中的用户研究单独评估。

1) 实验设置：在消融研究中，我们按如下方式配置了实验环境：

基线方法。我们将IoTGPT 与两种定制基线方法进行了比较：NoDecomp 和NoMem。NoDecomp 移除了任务分解，并直接根据用户指令推断所需的设备命令，采用了先前智能体 [8], [10], [11] 使用的仅推导设计。NoMem 保留任务分解，但禁用了任务记忆，使整个流程在不复用过去子任务知识的情况下运行。为确保公平比较，两种基线方法使用与IoTGPT相同的提示词。

数据集与指标。我们使用了与对比研究相同的数据集，并将STR作为主要准确率指标，同时测量延迟和成本。

流程。我们在冷启动和热启动阶段进行消融研究，并将 GPT-4o 用作所有智能体的骨干模型。在冷启动阶段，每个智能体都首次处理所有指令。在此阶段，IoTGPT和 NoDecomp 都会构建任务记忆，并即时复用可用知识。两者的关键区别在于粒度：IoTGPT以细粒度的子任务级别存储知识，而NoDecomp 由于缺少分解，仅在任务级别存储知识。在热启动阶段，每个智能体都使用改写后的数据集版本重新处理相同的任务，与对比研究中的做法一致。IoTGPT 和 NoDecomp 利用已构建的记忆，而NoMem 则针对每条指令从头开始执行推理流程。

2）准确率：图 7(a) 展示了各方法的 STR。我们通过比较IoTGPT 和 NoDecomp来分析任务分解的影响。在冷启动阶段，IoTGPT优于NoDecomp，这表明将任务分解为子任务能够提高推理准确率。这一绩效差距在热启动阶段进一步扩大。这主要是因为NoDecomp 依赖粗粒度的任务级记忆，当改写后的指令呈现出较大的语言变体时，往往无法检索到可复用的知识，因此 STR 仅提升了 1.37%。相比之下，IoTGPT利用细粒度的子任务级复用；即使因改写导致任务级匹配失败，它仍能复用已分解子任务的知识。这种细粒度复用使IoTGPT能够在任务共享相同推导子任务时，借助跨任务学到的知识，从而实现了 9.87% 的显著提升。这表明，任务分解不仅能提升推理绩效，还能与任务记忆协同作用，增强系统在语言变体下的稳健复用能力。

接下来，我们通过比较IoTGPT 和NoMem来考察任务记忆的作用。在冷启动阶段，IoTGPT 会根据成功执行构建任务记忆，并复用可用知识，而不是反复依赖大语言模型，从而减少命令生成出错的机会。相反，完全依赖大语言模型且不使用记忆的NoMem由于频繁出错，STR较低。值得注意的是，在热启动阶段，NoMem在改写数据集上的STR下降了1.31%，尽管它重新处理的任务在语义上与此前见过的任务相似。这一观察表明，任务记忆在减少命令生成错误方面发挥着重要作用。

3) 延迟与成本：图 7(b) 和 7(c) 分别展示了各方法的延迟和成本。比较 IoTGPT与 NoDecomp时，IoTGPT 在冷启动阶段会产生较高的开销。这一增长主要源于任务分解所需的提示中包含了详细的设备规格，这是提升准确率所必须做出的权衡。然而，在热启动阶段，IoTGPT通过利用任务记忆跳过冗余的大语言模型推理，实现了与 NoDecomp 相当的效率。相比之下，NoMem在两个阶段都始终表现出最高的延迟和成本，因为它不得不针对每条指令重复执行分解过程。这些结果表明，尽管任务分解会带来固有开销，但任务记忆能够显著减轻这一负担，从而确保系统效率。

IX. 用户研究

为补充第 VIII 节中的定量评估，我们开展了两项用户研究：*i)* 研究*1*：智能家居环境变化下的个性化，旨在考察IoTGPT 能否在设备配置发生变化时适应用户偏好；以及*ii)* 研究*2*：不同任务难度下的整体可用性

表 I 个性化研究所用BASE 和NEW 智能家居环境的设备配置。

	基础环境	新环境
温度	空调，恒温器，温湿度传感器	温湿度传感器，Fan
湿度	空调	Fan
亮度	灯光，百叶窗	灯光、百叶窗
噪声	TV, 吸尘器	TV, 吸尘器
空气质量	空调，空气质量传感器	智能窗户，空气质量传感器
安全	门锁	家庭摄像头

级别，该研究评估了IoTGPT 与基线系统相比，在不同复杂度的自动化任务中的可用性。

我们通过在线社区广告招募了15名参与者（11名男性、4名女性；平均年龄为= 28.1岁，范围为 = 24–32岁）。其中4名参与者此前使用过 Google Home 或 Samsung SmartThings 等物联网平台，14名参与者使用过包括 Google Assistant、Apple Siri 或 Samsung Bixby 在内的对话代理。每次实验持续60–80分钟，参与者获得10美元的报酬。用户研究符合本机构的机构审查委员会政策及知情同意书要求。

A. 研究*1*：自适应个性化评估 *ion*

我们开展了一项用户研究，以评估IoTGPT的个性化能力。与定量指标不同，生成的参数是否反映用户偏好本质上具有主观性。因此，我们依靠人工评估来检验IoTGPT 是否能随着设备可用性的变化，有效地对参数进行个性化设置。

1）实验设置：我们按照以下方式设置实验环境：

智能家居设置。 如表 I 所示，我们准备了两个环境：BASE 和NEW，每个环境都包含九种设备。在 EUPont本体中，每种设备都与特定的环境属性相关联。虽然两个环境共享若干设备，但它们也存在一些设备差异。例如，BASE包含空调、恒温器和智能门锁，而NEW则包含风扇、智能窗户和家庭摄像头。这一设置模拟了异构的智能家居配置，要求系统根据可用设备的功能调整用户偏好。

待比较方法。 我们比较了与先前实验相同的方法，即Sasha 和SAGE。对于IoTGPT，我们没有分别评估冷启动和热启动，因为是否存在任务记忆对个性化质量没有显著影响。所有方法均使用 GPT-4o 作为骨干模型。

数据集。 如表 II 所示，我们从第 VIII 节介绍的数据集中选取了 8 项任务。我们根据个性化难度选择任务，个性化难度由需要考虑的环境属性数量决定：涉及的属性越多，难度越高。

流程。 为实现个性化，IoTGPT需要设备交互日志来构建偏好表。然而，由于在有限的实验时段内，实际上无法收集每位参与者的此类日志，我们使用 GPT-4o 随机生成了 100 条交互日志。IoTGPT从这些日志中提取了关于环境属性的偏好表，并在研究期间使用这些偏好表。实验开始前，研究人员向参与者说明了研究中使用的偏好表，只有在参与者确认理解后，他们才参加实验。作为比较，SAGE 和 Sasha获得了相同的交互日志，但处理方式不同。SAGE 完全依靠这些日志来推断设备特定偏好，符合其原始设计；而原本不具备个性化功能的Sasha，则通过将这些历史记录纳入提示词进行了扩展。

本实验采用被试内设计。为尽量减少顺序效应，每位参与者看到的系统呈现顺序均经过随机化。首先，在每次实验中，所有系统都在BASE下从头开始处理 8 项任务。随后，它们在NEW 下使用经过修改的设备配置重新执行相同的任务。每个阶段结束后，参与者使用 7 点李克特量表从三个方面对输出进行评分（见表 III）：感知准确率（Q1）、设备级个性化（Q2）和参数级个性化（Q3）。Q1 反映参与者对正确性的主观感知，不受真实标注影响，因此可补充第 VIII 节报告的客观准确率。Q2 和 Q3 评估各系统在设备和参数层面根据用户偏好调整输出的效果。系统身份经过隐藏，以避免造成偏差；后续访谈则记录了参与者的判断依据。

2) 结果：表 III 总结了三个系统在BASE 和NEW环境下的参与者评分。在 Q1 中，IoTGPT 在两种环境下的评分均为最高，其次是SAGE和Sasha。这一趋势与第 VIII 节的客观准确率结果一致，表明参与者也认为IoTGPT 是准确率最高的系统。例如，P2 指出，IoTGPT“清楚地列出了完成任务所需的所有命令”；而 P11 则批评Sasha 产生了“许多毫无意义且准确率很低的输出。”

在 Q2 和 Q3 中，参与者始终给予 IoTGPT 最高评价，强调它“在两种环境中都尽可能充分地体现了用户偏好”（P4）。相比之下，SAGE被认为效果有限：P12 指出，它“选择设备的表现还算合理，但参数值偏差很大”，并进一步指出，“在 *NEW* 环境中，它似乎只满足了多项偏好中的一部分。”Sasha的评分最低，P15 评论说，它“忽视了用户偏好，反而自行判断要提供哪些服务。”这些结果表明，即使设备配置发生变化，IoTGPT仍始终实现了最出色的个性化。

为考察三种方法在 Q1、Q2 和 Q3 得分上的统计显著性，我们进行了 Friedman 检验。在 BASE 中，三个问题均观察到显著差异（Q1：$\chi^2(2) = 24.03$, $p < 0.001$，Q2：$\chi^2(2) = 24.4$, $p < 0.001$，Q3：$\chi^2(2) = 24.4$, $p < 0.001$）。为考察两两差异，我们进行了事后威尔科克森符号秩检验，并采用 Holm–Bonferroni 校正。分析表明，各方法两两之间均存在显著差异（均为 $p < 0.05$）。同样，在 NEW 中，三个问题均发现显著差异（Q1：$\chi^2(2) = 24.4$,$p < 0.001$，Q2：$\chi^2(2) = 24.4$, $p < 0.001$，Q3：$\chi^2(2) = 24.4$,

表 II 个性化研究中使用的任务集。任务根据所涉及的环境属性数量分为三个难度级别。B = 亮度，T = 温度，H = 湿度，N = 噪声，A = 空气质量，S = 安全。

难度级别（属性数量）	**任务（用户指令）**	**环境属性**					
		B	T	H	N	A	S
简单（1–2）	帮我凉快一下		✓				
	把氛围调得更能让我振作起来	✓			✓		
	这里有点太吵了				✓		
中等（3–4）	营造良好的学习环境	✓	✓	✓	✓		
	将房间布置成轻松的氛围	✓	✓		✓		
	我累了——把房间布置得舒适些，让我可以休息	✓	✓		✓		
困难（5–6）	营造舒适的居家氛围	✓	✓	✓	✓	✓	
	为我营造最佳睡眠环境	✓	✓	✓	✓	✓	✓

表 III 研究1的调查结果。每项评分均在任务完成后使用 7 点李克特量表收集。数值表示均值，括号内为标准差。

	基础环境			**新环境**		
	Sasha	SAGE	IoTGPT	Sasha	SAGE	IoTGPT
问题 1. 是否包含了所有必要的设备命令（无论偏好如何）？	2.47 ± 0.21	3.68 ± 0.13	5.95 ± 0.05	1.93 ± 0.19	3.90 ± 0.13	5.78 ± 0.05
Q2. 所选设备是否适合 合乐偏好?	2.44 ± 0.23	3.52 ± 0.14	6.28 ± 0.04	1.93 ± 0.19	3.72 ± 0.14	5.63 ± 0.09
Q3. 设备设置是否根据偏好进行了调整?	2.20 ± 0.21	3.04 ± 0.11	6.02 ± 0.05	1.93 ± 0.19	3.18 ± 0.14	5.57 ± 0.10

$p < 0.001$）。事后威尔科克森符号秩检验显示，各方法两两之间均存在显著差异（均为$p < 0.05$），这表明 IoTGPT 在偏好适应方面显著优于两个基线系统。

B. 研究二：整体可用性评估

我们开展了一项用户研究，以评估 IoTGPT 在真实智能家居场景中的可用性和交互体验。我们将用户使用 IoTGPT 的体验与基线方法进行了比较，重点考察我们的设计选择是否能带来切实的效益。具体而言，我们试图回答以下两个研究问题：*i) IoTGPT* 的分解流程是否能帮助用户实现更有效的自动化结果？以及 *ii) IoTGPT* 的分层任务记忆是否能提供可靠且快速的响应？

1）实验设置：我们按如下方式设置了实验环境：

智能家居设置。研究中使用的智能家居环境由七种常见的物联网设备组成：存在传感器、灯光、门锁、百叶窗、空调、冰箱和电视。

待比较方法。我们将IoTGPT与两种基线方法进行了比较：SmartThings 应用和SAGE。我们将 SmartThings 应用（在 Apple iPad 4 上测试）作为实用参考，用于了解用户如今如何通过主流图形界面执行物联网命令。我们选择SAGE作为以往基于大语言模型的智能体中最强的研究基线，因为在我们的比较评估中，它的准确率和个性化表现始终优于Sasha 。我们在两种条件下评估IoTGPT ，以衡量任务记忆对可用性的影响：IoTGPT-Cold从空白记忆开始，代表首次使用体验；IoTGPT-Warm则预先加载了根据第八节的 97 个任务构建的任务记忆，代表记忆积累后持续使用的情况。两种条件使用相同的偏好表，这些偏好表根据第九节 A 小节中的用户画像推导得出，因为按参与者收集交互历史并不实际。请注意，所有智能体均使用 GPT-4o 作为骨干模型。

流程。在每次实验开始时，我们都会向每位参与者介绍智能家居技术和对话代理。我们还解释了研究中使用的智能家居配置，以及每种方法的使用方式；随后进行简短教程，让参与者在正式任务开始前练习使用这些系统。

随后，参与者被要求使用四种方法（SmartThings 应用、SAGE、IoTGPT-Cold 和IoTGPT-Warm）完成三个难度递增的任务场景（简单、中等和困难）。所有参与者面对的场景均相同。对于每个场景，我们都提供了场景描述和示例指令（见表 IV），同时允许参与者采用不同的措辞表达指令。所有任务均为触发-动作规则，并按难度递增的顺序呈现。为减轻不同方法之间的顺序效应，我们在参与者之间对方法顺序进行了平衡。

每完成一个场景后，参与者都填写 NASA-TLX问卷，以评估感知工作负荷，并接受访谈以即时反馈。四种方法下的三个场景均重复这一流程。最后，完成所有任务后，参与者使用系统可用性量表（SUS）评估四种方法的可用性。整个环节以访谈作结，参与者在访谈中回顾了每种方法的优缺点。

2）结果：现在，我们将介绍第二项用户研究的结果。该研究比较了四种方法在不同难度任务中的命令生成时间、感知工作负荷和可用性。

任务完成时间。我们测量了三种场景下完成每项自动化任务所需的时间。为考虑不同的交互模式，我们针对每种方法对该指标进行了不同定义：对于 SmartThings 应用，

表 IV 按难度级别（简单、中等、困难）分组的可用性研究任务场景

难度级别	场景描述	示例指令
Easy（≥ 2 个设备，明确指定）	凌晨3点，你口渴醒来，悄悄地走进厨房。你打开冰箱门，但房间太暗了，你找不到一杯水。你心想，要是每次打开冰箱门时，餐厅灯都能自动亮起来就好了。	“打开餐厅的灯 在我打开冰箱时 ”
中等（≥ 4 设备，未指定）	每天晚上，你都会以看电视电影的方式结束一天。为了更专注地享受观影体验，你通常会布置周围环境。然而，每次都重复这样的准备工作，开始让你觉得越来越麻烦。	“看电视时，我希望家里能像电影院一样”
Hard（≥ 6 设备，未指定）	漫长而疲惫的一天结束后，你回到卧室，为睡眠营造舒适的环境。然而，每晚都重复同样的流程，在一天结束时亲自做好准备，一天开始变得越来越乏味、令人烦恼。	“让房间变得舒适，方便我晚上入睡”

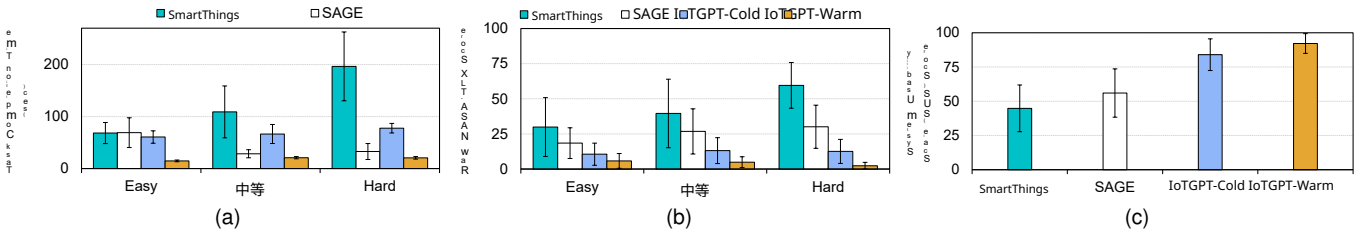


图 8. (a) 各场景的平均任务完成时间。(b) 各场景的 NASA-TLX 平均原始得分。(c) 各方法的平均可用性得分。每项得分均采用 100 分制评估，误差线表示标准误。

该指标表示从开始设置自动化到点击“保存”按钮所需的人工交互时间，点击该按钮后，配置的自动化将应用于智能家居环境。对于对话代理（SAGE 和 IoTGPT），该指标表示从参与者提交自然语言指令到生成的命令完全应用于智能家居环境的端到端延迟。

如图 8(a) 所示，除简单场景外，SmartThings 应用所需时间始终最长。此外，其完成时间会随着任务难度增加而急剧上升：在高难度场景中，所需时间比简单场景高出 187.7%。这反映了手动指定每个设备动作所带来的额外负担，而随着任务变得更加复杂，这种方式难以扩展。与此同时，SAGE 在简单场景中耗时最长，但在中等和高难度场景中则呈现相反趋势，耗时有所减少。这一反直觉的结果源于 SAGE 将触发条件-动作任务误解为更简单的直接控制命令，从而缩短了任务完成时间，却导致自动化设置不准确[2]。这凸显了过度依赖大语言模型的风险。

IoTGPT-Cold 随着任务复杂度增加，耗时也会增加，但增幅低于 SmartThings（从简单场景到高难度场景增加了 28.1%）。相比之下，IoTGPT-Warm利用任务记忆，在所有场景中都大幅缩短了耗时；与IoTGPT-Cold 相比，在简单、中等和高难度场景中的耗时分别减少了 75.67%、68.54% 和 73.46%。这些结果凸显了重复利用先前学到的子任务所带来的效率提升。

我们使用重复测量方差分析来分析不同方法在任务完成时间上的差异，并在事后检验中应用邦费罗尼校正，以控制多重比较。方差分析显示，方法对完成时间有显著影响适用于所有场景（简单:$F(3,\ 42)\ =\ 31.84,\ p\ <\ 0.001$; 中等：$F(3,\ 42)\ =\ 32.59, p < 0.001$; 困难：$F(3,\ 42)\ =\ 79.19,\ p < 0.001$）。

在简单场景中，事后比较显示，IoTGPT-Warm显著快于 SmartThings 应用、SAGE和IoTGPT-Cold （均为 $p\ <\ 0.001$）。其他配对比较均未达到显著性。在中等难度和高难度场景中，由于SAGE误解任务并导致输出无效，因此未将其纳入统计比较。在中等难度场景中，IoTGPT-Warm的表现优于 SmartThings 应用和IoTGPT-Cold （均为 $p\ <\ 0.001$），而 SmartThings 应用与IoTGPT-Cold之间未发现显著差异。在高难度场景中，所有配对比较均达到显著性（均为 $p\ <\ 0.05$）。在所有场景中，IoTGPT-Warm始终耗时最短，而且随着任务复杂度增加，其优势愈发明显。

感知任务负荷。图 8(b) 展示了各场景的 NASA-TLX 原始平均得分。SmartThings 应用始终产生最高的负荷，其次是SAGE。SAGE的得分偏高，主要是因为任务结果不佳：它常常无法生成令人满意的命令，导致绩效（表示感知到的结果较差）和挫败感评分更高。相比之下，IoTGPT-Cold 大幅降低了负荷，而IoTGPT-Warm 则进一步将负荷降至所有场景中的最低水平。这表明IoTGPT能够有效降低认知努力，让用户在处理复杂任务时不会感到不堪重负。

弗里德曼检验显示，在所有场景中，方法均产生显著影响（简单：$\chi^2(3) = 18.69$；中等：$\chi^2(3) = 26.49$；困难：$\chi^2(3)\ =\ 37.54$；全部 $p\ <\ 0.001$）。采用 Holm–Bonferroni 校正的事后 Wilcoxon 符号秩比较结果如下。在简单场景中，IoTGPT-暖 的工作负荷显著低于 SmartThings 应用（$p = 0.007$）和SAGE （$p = 0.009$），而IoTGPT-Cold与IoTGPT-Warm 之间的差异不显著

[2]具体而言，SAGE 在中等难度和高难度场景下，未能为参与者的所有任务生成正确输出；而两个 IoTGPT 版本则始终能在所有场景中生成正确的物联网命令。

（$p = 0.129$）。IoTGPT-Cold的工作负荷也显著低于 SmartThings 应用（$p = 0.017$），但与SAGE 相比则无显著差异（$p = 0.0905$）。在中等场景中，IoTGPT-Warm的工作负荷再次显著低于 SmartThings 应用（$p = 0.005$）、SAGE （$p < 0.001$）和IoTGPT-Cold （p =0.016）。IoTGPT-Cold的工作负荷也显著低于 SmartThings 应用（$p = 0.002$）和SAGE（$p = 0.025$）。在高难度场景中，所有两两比较结果均显著（全部 $p < 0.05$）。

可用性。 图 8(c) 展示了各方法的平均 SUS 评分。SmartThings 应用的评分最低，其次是 SAGE，而两个版本的 IoTGPT得分都明显更高。具体而言，IoTGPT-Cold 已达到优秀 水平，而IoTGPT-Warm 则进一步达到最佳想象类别。这些结果凸显了IoTGPT 在感知可用性方面明显优于两种基线方法。

弗里德曼检验显示，不同方法对可用性的影响显著（$\chi^2(3) = 32.05, p < 0.001$）。事后 Wilcoxon 符号秩比较证实，IoTGPT-Warm的可用性评分显著高于 SmartThings 应用（$p < 0.001$）、SAGE （$p = 0.003$）和 IoTGPT-Cold （$p = 0.037$）。IoTGPT-Cold 的可用性也显著高于 SmartThings 应用（$p = 0.001$）和 SAGE （$p < 0.001$）。

C. 总体关键启示

两项用户研究的发现带来了四项关于智能家居自动化系统设计的关键启示。

手动界面与对话式界面。 随着任务复杂度增加，参与者始终认为 SmartThings 应用使用起来颇为繁琐。例如，P3 提到：“现有平台感觉过于复杂，也很难用，因为可供选择的项目太多了。”同样，P13 评论道：“我觉得这会让年长用户难以使用。”面对更复杂的任务时，参与者难以确信自己的操作是否正确，也不清楚该如何进行；P6 承认：“我不确定自己设置得对不对”，P5 则表示：“我不知道该怎么做，才能让房子有电影院的感觉。”综合来看，这些评价凸显了手动可视化界面的可扩展性局限。

相比之下，人们认为对话代理更加自然、高效，尤其是在处理复杂任务时。参与者称赞IoTGPT：P1 指出：“只需一个语音命令就能控制多个设备，非常直观方便。”P7 补充道：“系统能推荐我自己都想不到的设备控制方式，这令人印象深刻。”另一位参与者强调了IoTGPT 处理模糊命令的能力：P9 回忆道：“它能自然而然地把‘弄得像电影院一样’这样模糊的要求具体化并执行，这一点很棒。就连我自己也得费心琢磨，这对我来说到底意味着什么。”这些发现凸显了自然语言交互对智能家居自动化的价值。

过度依赖大语言模型的风险。 尽管现有的基于大语言模型的智能体支持自然交互，但它们过度依赖大语言模型，导致结果不稳定、响应速度较慢。尤其是SAGE，它会针对每项任务从头开始重新规划推理模块，并反复查询大语言模型直至任务完成，因此高度依赖大语言模型。这不仅延长了执行时间，也增加了出错的可能性当任务变得更加复杂或模糊时。参与者直接指出了这些缺点。P2 表示：“即使是简单的任务，耗时也比我预想的更长。”P10 指出：“它忽略了我在命令中指定的条件（触发条件），只生成动作部分，因此在现实生活中不实用。”P8 分享了一个更严重的担忧：“系统突然把电视音量调到了 *100*。对于这种行为，应该设置一道防护措施。”总体而言，这些发现表明，需要采用互补机制（例如任务记忆），以减少对大语言模型的重复依赖，并确保更可靠的绩效。

冷启动与热启动的可用性对比。我们的研究结果表明，即使在不存在既有任务记忆的冷启动条件下，IoTGPT 仍具有很高的可用性。参与者依然能够以相对较低的工作负荷和较高的满意度完成复杂的自动化任务，这表明即使没有事先学习，系统的分解流程也已能带来切实的益处。然而，热启动版本进一步增强了这些优势：通过复用积累的任务记忆，它缩短了命令生成时间、降低了工作负荷，并提升了整体可用性。这说明，尽管IoTGPT开箱即用、易于使用，但随着任务记忆不断增长，其可用性也会逐步提升，为长期采用提供了清晰的路径。参与者的反馈也印证了这些观察结果。P9指出：“即使没有事先学习，它仍然运行良好，使用起来也很方便。”P12强调了这种逐步改善：“早期版本（IoTGPT-Cold）比我预期的耗时更长，但后续版本（IoTGPT-Warm）很快就给出了想要的结果，令人非常满意。”同样，P10补充道：“早期版本已经足够好用，但经过学习的版本速度快到足以处理即时命令，让它显得实用得多。”

个性化的重要性。我们的研究表明，个性化是智能家居自动化的核心需求。参与者重视能够根据其偏好和情境定制命令的系统，并指出这种支持让自动化更实用、更值得信赖。例如，P10 解释道：“多个设备能够按照给定的偏好进行调整，而我无需手动配置，这很方便。”同样，P7 表示：“系统似乎甚至能推荐一些有用的设备使用模式，而这些模式连我自己都没有意识到。”除静态个性化外，研究1的结果还凸显了自适应性的重要性：当设备配置发生变化时，在环境属性层面映射偏好的系统能够更有效地保留用户偏好。正如 P4 所说：“即使设备有所不同，系统仍然能够体现我的偏好，这令人印象深刻。”总之，这些发现强调，个性化——尤其是自适应个性化——对于在动态家庭环境中维持可用性至关重要。

X. 局限性与未来工作

冲突的用户偏好的困境。当不同环境属性下的用户偏好相互冲突时，个性化可能变得模糊。例如，一位用户 who

然而，如果用户更看重安静而非温度，这可能导致个性化效果不佳。为解决这一问题，未来的工作应扩展本体，以表示设备的副作用，并细化偏好粒度，从而使系统能够更有效地解决此类冲突。

本地推理的必要性。尽管IoTGPT通过任务记忆降低了命令生成延迟，但仍有一些参与者觉得响应有所延迟。例如，P4 表示：“虽然执行时间经过优化，但感觉还是很长。”除了绩效方面的问题，人们也对数据隐私表示担忧。正如 P6 所指出的：“自然语言命令可能会无意中包含私人信息，如果由云端大语言模型处理，这些信息可能会泄露。”这些观察表明，未来需要致力于实现本地推理。开发面向智能家居自动化的轻量级大语言模型，既能降低延迟，也能缓解隐私风险，使此类系统在日常使用中更加实用、可靠。

上下文感知的局限性。IoTGPT的一项局限在于，它仅根据用户指令推断上下文，无法充分捕捉情境因素。例如，将活动上下文（如通过视觉语言模型分析家庭摄像头图像，识别用户是在休息还是锻炼）与环境状态上下文（如通过传感器测量的温度、湿度或噪声）相结合，可以更准确地细化物联网命令。然而，纳入此类多模态上下文可能会导致任务记忆过度碎片化，使个性化知识难以复用。因此，未来工作应探索抽象技术，例如对上下文进行聚类或层次化表示，以捕捉多样情境，同时保持可复用性。

从人工检查到内置安全机制。在我们当前的原型中，筛查不安全或幻觉命令的责任落在用户身上，用户必须检查生成的自动化操作，并在必要时提供纠正性反馈。虽然这种方法能避免即时风险，但会增加用户的负担并降低可用性。为解决这一局限，未来的工作应探索自动化安全机制，在命令执行前主动验证或调整命令。例如，如果系统错误地生成了将电视音量调至 100 的命令，内置防护机制便可自动将数值限制在安全阈值（例如 50），而无需用户干预。这类验证和故障安全措施能够在减轻用户负担的同时提高可靠性，使IoTGPT 只需极少监督便能实现安全的自动化。

长期用户体验。我们的评估仅限于短期使用，因为参与者在用户研究期间与IoTGPT 互动了约 60 分钟。虽然这些环节为了解可用性和个性化提供了有用的见解，但仍无法全面反映用户在更长时间的日常生活中会如何适应或依赖IoTGPT。未来的研究应开展纵向研究，考察持续使用如何影响现实智能家居环境中的用户信任、系统适应性和整体满意度。

XI. 结论

我们介绍了IoTGPT，这是一种基于大语言模型的智能家居代理，利用任务分解和任务记忆，以可靠、高效且个性化的方式生成物联网命令。IoTGPT 将结构化推理与记忆重用相结合，以降低物联网命令生成的不可靠性、延迟和成本，同时通过设备无关的偏好建模支持自适应个性化。我们在性能基准测试和用户研究中的评估表明，与基线方法相比，该系统在准确率、效率和可用性方面均有所提升。我们认为，IoTGPT是迈向更可靠、以用户为中心的人工智能代理的切实一步，其影响不仅限于智能家居，也延伸至其他需要可靠且自适应自动化的广泛情境。

参考文献

[1] T. Thakur，《物联网统计数据 2025年：设备、安全与普及》，https://sqmagazine.co.uk/internet-of-things-statistics/，2025年，访问日期：2025-07-30。[2] L. Bainbridge，《自动化的悖论》，*IFAC*会议论文集，第15卷，第6期，第129–135页，1982年。[3] 三星，《SmartThings》，https://www.samsung.com/sec/smartthings/ app/，2025年，访问日期：2025-07-23。[4] Google，《Google Home》，https://home.google.com/welcome/，2025年，访问日期：2025-07-23。[5] 苹果，《Apple Home》，https://www.apple.com/kr/home-app/，2025年，访问日期：2025-07-23。[6] Google，《Hey Google》，https://assistant.google.com/，2025年，访问日期：2025-07-10。[7] 苹果，《在所有苹果设备上使用 Siri》，https://support.apple.com/en-us/105020，2025年，访问日期：2025-07-10。[8] E. King、H. Yu、S. Lee 和 C. Julien，《Sasha：利用大语言模型在智能家居中进行创造性目标导向推理》，《*ACM*交互、移动、可穿戴与泛在技术汇刊》，第8卷，第1期，第1–38页，2024年。[9] D. Rivkin、F. Hogan、A. Feriani、A. Konar、A. Sigal 和 X. Liu，《通过自主大语言模型代理实现人工智能物联网智能家居》，《*IEEE*物联网期刊》，第12卷，第3期，第2458–2472页，2025年。[10] Y. Shi、X. Liu、C. Yu、T. Yang、C. Gao、C. Liang 和 Y. Shi，《弥合智能家居中自然用户表达与复杂自动化编程之间的鸿沟》，https://arxiv.org/abs/2408.12687，2024年，arXiv:2408.12687。[11] S. Gallo、F. Patern`o 和 A. Malizia，《利用大语言模型创建自动化的对话代理》，《个人与泛在计算》，第28卷，第931–946页，2024年。[12] T. Cal`o 和 L. De Russis，《通过多模态命令消歧增强智能家居交互》，第28卷，1000，2024年 6，985no. p. – 。[13] C. G. Correa、M. K. Ho、F. Callaway、N. D. Daw 和 T. L. Griffiths，《人类通过权衡效用与计算成本来分解任务》，《公共科学图书馆计算生物学》，第19卷，第6期，第e1011087 页，2023年。[14] M. L¨ovd´en、B. Garz´on 和 U. Lindenberger，《人类技能学习：扩展、探索、选择与细化》，《行为科学当前观点》，第36卷，第163–168页，2020年。[15] X. Chen、S. Chen、Z. Jin、H. Bian、Z. Chen 和 H. Li，《表达智能家居需求：终端用户最喜欢哪种方式》，《*ACM*计算机与人类交互汇刊》，第32卷，38，2025年 16，1no. pp. – 。[16] T.-H. K. Huang、A. Azaria 和 J. P. Bigham，《InstructableCrowd：通过与众包群体对话创建如果—那么规则》，载于*2016*年*CHI*人机交互大会论文集（*CHI EA’16*），2016年。[17] G. Campagna、R. Ramesh、S. Xu、M. Fischer 和 M. S. Lam，《Almond：一种开放、众包、保护隐私且可编程的虚拟助手架构》，载于国际万维网大会委员会（*IW3C2*），2017年。

[18] J. Jaihar、N. Lingayat、P. S. Vijaybhai、G. Venkatesh 和 K. P. Upla，《使用机器学习算法实现智能家居自动化》，载于新兴技术国际会议（INCET），2020年。[19] S. Kim 和 I.-Y. Ko，《在物联网环境中修改服务混搭的对话式方法》，载于CHI人机交互大会（CHI '22），2022年。[20] G. Simone 和 P. Fabio，《用于创建灵活日常自动化的对话代理》，载于2022年高级可视化界面国际会议论文集（AVI 2022），2022年。[21] F. Corno、L. D. Russis 和 A. M. Roffarello，《从用户意图到物联网中的如果—那么规则》，信息系统汇刊（TOIS），第39卷，第53期，第1–33页，2021年。[22] S. Lee、J. Choi、J. Lee、M. H. Wasi、H. Choi、S. Y. Ko、S. Oh 和 I. Shin，《MobileGPT：借助类人应用记忆增强大语言模型，实现移动任务自动化》，载于第30届年度移动计算与网络国际会议论文集，2024年。[23] H. Wen、Y. Li、G. Liu、S. Zhao、T. Yu、T. J.-J. Li、S. Jiang、Y. Liu、Y. Zhang 和 Y. Liu，《AutoDroid：由大语言模型驱动的 Android 任务自动化》，载于第24届年度移动计算与网络国际会议论文集，2024年。[24] C. Zhang、Z. Yang、J. Liu、Y. Li、Y. Han、X. Chen、Z. Huang、B. Fu 和 G. Yu，《AppAgent：作为智能手机用户的多模态代理》，载于2025年CHI人机交互大会论文集，2025年。[25] J. Lee、D. Lee、C. Choi、Y. Im、J. Wi、K. Heo、S. Oh、S. Lee 和 I. Shin，《通过基于逻辑的动作验证保障移动图形用户界面代理的安全》，arXiv预印本 arXiv:2503.18492，2025年。[26] H. Wen、S. Tian、B. Pavlov、W. Du、Y. Li、G. Chang、S. Zhao、J. Liu、Y. Liu、Y.-Q. Zhang 等，《AutoDroid-v2：通过代码生成提升基于小语言模型的图形用户界面代理》，arXiv预印本 arXiv:2412.18116，2024年。[27] R. Nakano、J. Hilton、S. Balaji、J. Wu、L. Ouyang、C. Kim、C. Hesse、S. Jain、V. Kosaraju、W. Saunders 等，《WebGPT：借助人类反馈进行浏览器辅助问答》，arXiv预印本 arXiv:2112.09332，2021年。[28] S. Zhou、F. F. Xu、H. Zhu、X. Zhou、R. Lo、A. Sridhar、X. Cheng、T. Ou、Y. Bisk、D. Fried、U. Alon 和 G. Neubig，《WebArena：用于构建自主代理的真实网页环境》，2024年，arXiv:2307.13854。[29] W. Hong、W. Wang、Q. Lv、J. Xu、W. Yu、J. Ji、Y. Wang、Z. Wang、Y. Dong、M. Ding 等，《CogAgent：面向图形用户界面代理的视觉语言模型》，载于IEEE/CVF计算机视觉与模式识别会议论文集，2024年，第14 281–14 290页。[30] A. Tomar、H. Liang、I. Bhattacharya、N. Larios 和 F. Car-bone，《Cybernaut：迈向可靠的网页自动化》，arXiv预印本 arXiv:2508.16688，2025年。[31] D. Gao、L. Ji、Z. Bai、M. Ouyang、P. Li、D. Mao、Q. Wu、W. Zhang、P. Wang、X. Guo、H. Wang、L. Zhou 和 M. Z. Shou，《Assist-gui：面向任务的桌面图形用户界面自动化》，2024年，arXiv:2312.13108。[32] R. Niu、J. Li、S. Wang、Y. Fu、X. Hu、X. Leng、H. Kong、Y. Chang 和 Q. Wang，《ScreenAgent：由视觉语言模型驱动的计算机控制代理》，arXiv预印本 arXiv:2402.07945，2024年。[33] C. Zhang、H. Huang、C. Ni、J. Mu、S. Qin、S. He、L. Wang、F. Yang、P. Zhao、C. Du 等，《UFO2：桌面代理操作系统》，arXiv预印本 arXiv:2504.14603，2025年。[34] D. Zhao、L. Ma、S. Wang、M. Wang 和 Z. Lv，《COLA：面向 Windows 用户界面任务自动化的可扩展多代理框架》，arXiv预印本 arXiv:2503.09263，2025年。[35] 三星，《SmartThings开发者平台》，https://developer.smartthings.com/docs/api/public，2025年，访问日期：2025-09-03。[36] F. Corno、L. De Russis 和 A. M. Roffarello，《面向物联网终端用户开发的高层次方法》，载于2017年CHI人机交互大会计算系统中的人为因素扩展摘要论文集，2017年。[37] OpenAI，《你好，GPT-4o》，https://openai.com/ko-KR/index/hello-gpt-4o/，2025年，访问日期：2025-07-11。[38] ——，《向量嵌入》，https://platform.openai.com/docs/guides/ embeddings，2025年，访问日期：2025-08-29。[39] 三星，《SmartThings Edge Driver》，https://developer.smartthings.com/ docs/devices/hub-connected/lan，2025年，访问日期：2025-08-28。[40] D. Rivkin，《SAGE》，https://github.com/SAIC-MONTREAL/SAGE，2025年，访问日期：2025-09-05。[41] OpenAI，《推出 GPT-5.2》，https://openai.com/index/introducing- gpt-5-2/，2025年，访问日期：2025-12-29。[42] Google，《Gemini 3 flash》，https://ai.google.dev/gemini-api/docs/models? hl=ko#gemini-3-flash，2025年，访问日期：2025-12-29。[43] Anthropic，《推出 Claude Opus 4.5》，https://www.anthropic.com/ news/claude-opus-4-5，2025年，访问日期：2025-12-29。