计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202440380
61(11): 2706−2720, 2024

面向大语言模型驱动的智能体的计划复用机制

李国鹏　吴瑞骐　谈海生　陈国良
（中国科学技术大学计算机科学与技术学院　合肥　230027）
（guopengli@mail.ustc.edu.cn）

A Plan Reuse Mechanism for LLM-Driven Agent

Li Guopeng, Wu Ruiqi, Tan Haisheng, and Chen Guoliang
（*School of Computer Science and Technology*, *University of Science and Technology of China*, *Hefei* 230027）

Abstract　Integrating large language models (LLMs) into personal assistants, like Xiao Ai and Blue Heart V, effectively enhances their ability to interact with humans, solve complex tasks, and manage IoT devices. Such assistants are also termed LLM-driven agents. Upon receiving user requests, the LLM-driven agent generates plans using an LLM, executes these plans through various tools, and then returns the response to the user. During this process, the latency for generating a plan with an LLM can reach tens of seconds, significantly degrading user experience. Real-world dataset analysis shows that about 30% of the requests received by LLM-driven agents are identical or similar, which allows the reuse of previously generated plans to reduce latency. However, it is difficult to accurately define the similarity between the request texts received by the LLM-driven agent through directly evaluating the original request texts. Moreover, the diverse expressions of natural language and the unstructured format of plan texts make implementing plan reuse challenging. To address these issues, we present and implement a plan reuse mechanism for LLM-driven agents called AgentReuse. AgentReuse leverages the similarities and differences among requests' semantics and uses intent classification to evaluate the similarities between requests and enable the reuse of plans. Experimental results based on a real-world dataset demonstrate that AgentReuse achieves a 93% effective plan reuse rate, an $F1$ score of 0.971 8, and an accuracy of 0.945 9 in evaluating request similarities, reducing latency by 93.12% compared with baselines without using the reuse mechanism.

Key words　artificial intelligence of things; large language models (LLMs); agent; semantic cache; similarity evaluation

摘　要　将大语言模型集成到个人助手中（如小爱同学、蓝心小V等）能有效提升个人助手与人类交互、解决复杂问题、管理物联网设备等能力，这类助手也被称为大模型驱动的智能体，也可称其为智能体.智能体接收到用户请求后，首先调用大模型生成计划，之后调用各类工具执行计划并将响应返回给用户.上述过程中，智能体使用大模型生成计划的延迟可达数十秒，十分影响用户体验.对真实数据的分析显示，智能体接收到的请求中约有30%是相同或相似的，此类请求可复用先前生成的计划，以降低智能体响应延迟.然而，直接对请求原始文本进行相似度评估难以准确界定智能体接收到的请求文本间的相似性.此外，自然语言表达的多样性和大模型生成的非结构化计划文本导致难以对计划进行有效复用.针对上述

收稿日期：2024-05-31；**修回日期**：2024-07-11
基金项目：科技创新2030—“新一代人工智能”重大项目(2021ZD0110400)；国家自然科学基金重点项目(62132009)；中央高校基本科研业务费专项资金
This work was supported by the 2030 National Key AI Program of China (2021ZD0110400), the Key Program of the National Natural Science Foundation of China (62132009), and the Fundamental Research Funds for the Central Universities.
通信作者：谈海生(hstan@ustc.edu.cn)

问题，提出并实现了面向大模型驱动的智能体的计划复用机制 AgentReuse，通过利用请求文本间语义的相似性和差异性，采用基于意图分类的方法来界定请求间的相似性并实现计划复用. 基于真实数据集的实验结果表明，AgentReuse 对计划的有效复用率为 93%，对请求进行相似性界定的 $F1$ 分数为 0.971 8，准确率为 0.945 9，与不采用复用机制相比，可减少 93.12% 的延迟.

关键词 智能物联网；大语言模型；智能体；语义缓存；相似度评估

中图法分类号 TP393

物联网是形成新质生产力的重要阵地和新一代信息技术的重要组成部分[1-2]. 近年来，随着物联网设备数量的增长以及人工智能、边缘计算等技术的兴起，智能物联网作为未来物联网发展的趋势得到了广泛关注[3]. 个人助手，如小爱同学[4]、华为小艺[5]等，作为典型的智能物联网应用，通过简化用户控制物联网设备的过程，提升了物联网设备在智慧家居、智慧交通等领域的实用性和集成度，从而增强了智能物联网系统的互动性和自动化水平. 近期，ChatGPT、LLaMA 3 和通义千问等大语言模型（large language models, LLMs）在理解和生成人类语言方面表现出了卓越的能力，为个人助手的发展带来了新的机遇[6-7]. 谷歌、小米和 VIVO 等厂商推出了基于大模型的个人助手，目标是利用大模型提升个人助手与人类交互、完成复杂任务、高效管理和控制应用程序及物联网等各类设备的能力，这类助手被称为大模型驱动的个人智能体（personal LLM-driven agent）[8]，也可以称为大模型驱动的智能体（LLM-driven agent），本文中也简称为智能体.

大模型驱动的智能体，以大模型为核心控制器，辅以规划、记忆、使用工具等能力[9]，通过协调不同的能力，完成用户提交给智能体的任务. 当智能体接收到用户提交的任务请求后，首先利用大模型的规划能力生成计划，即将复杂任务分解为一系列简单的子任务，之后通过调用 API、代码解释器、搜索引擎等工具，结合大模型已拥有的记忆（如用户个人偏好等），执行子任务以完成整个任务. 例如，当用户提交的任务请求为“预定后天合肥到北京的票”时，智能体利用大模型生成的计划为“1. 查询后天的日期 2. 查询天气 3. 查询合肥到北京的飞机票班次、价格等信息 4. 查询合肥到北京的火车票班次、价格等信息 5. 比较飞机票和火车票的价格 6. 查询用户个人日程 7. 结合上述信息和用户个人偏好得出推荐出行方案.”得出计划后，智能体通过调用手机上的应用程序或开放式的 API 完成子任务 1, 2, 3, 4, 6，通过代码解释器完成子任务 5，最后，对于子任务 7，结合天气、价格信息、个人日程和记忆中存储的个人偏好得到后天的出行方案.

在例子中，我们可以看到，集成了大模型能力的智能体能结合用户个人需求更好地完成任务[10-11]. 但使用大模型的计算和延迟代价是不可忽视的，使用 AutoGen[12] 作为智能体框架，OpenAI 的 GPT-4 API 作为使用的大模型，输入的任务为“预定后天合肥到北京的票”，返回计划的延迟约为 25 s，消耗的 token 数为 2015，OpenAI 的收费是 0. 073 美元（约 0.53 元人民币）. 在 25 s 的延迟中，大模型生成计划的延迟约 24 s，其余为网络延迟，这是因为大模型以自回归的方式生成计划，计划文本普遍较长，因此生成计划的延迟较长. 25 s 的计划生成延迟，再加上后续执行计划的延迟，十分影响用户体验. 本文对文献 [13] 中的个人助手数据集的分析和文献 [14] 均表明，在大模型驱动的智能体中，约有 30% 的任务请求在语义上是相同或相似的. 因此，将智能体使用大模型生成的计划缓存起来，在后续到达相同或相似的请求时，复用之前的计划，是减少大模型驱动的智能体的响应延迟的有效方法[15].

文献 [16] 研究了面向大模型本身的响应的缓存（cache）和复用（reuse）方法，通过将请求文本进行向量化（embedding）比较不同请求文本间的语义相似度来判断后续的请求是否可以复用之前的请求的响应. 如：“请解释什么是智能物联网”与“智能物联网是什么”在语义上是相似的，可以直接复用大模型对前者的响应，而不是让大模型再生成一次对于智能物联网的介绍，以达到减少响应延迟的目的. 但在大模型驱动的智能体处理用户请求的过程中，对于如下 2 个请求，请求 1：“预订后天合肥到北京的票”与请求 2：“定一下明天长沙到上海的票”，当直接采用文献 [16] 中的方法时，若请求 2 被判定为与请求 1 相似，在处理请求 2 时，直接返回请求 1 的响应，很明显，这会返回给用户错误的购票信息. 如果将请求 2 判定为与请求 1 不相似，则会产生约 25 s 的计划生成延迟. 事实上，通过观察我们不难发现，对于请求 1 和请求

2 来说, 虽然时间、出发地、目的地不同, 但任务的类型是相同的, 因此, 智能体产生的计划步骤应是相同的, 只是具体使用的关键参数“时间、出发地、目的地”不同, 即请求 2 与请求 1 应该被判定为相似, 且不应该直接为请求 2 返回请求 1 的响应, 而应是请求 2 去复用大模型为请求 1 生成的计划.

本文面向大模型驱动的智能体的计划复用机制展开研究, 现有研究工作主要面向大模型本身响应的缓存与复用机制展开研究. 本文研究主要面临 3 个挑战:

挑战 1: 如何界定请求是否相似. 如前文所述, 在面向大模型的响应的缓存和复用方法中, 可以通过将请求原始文本进行向量化来界定请求间的语义相似度. 但在面向大模型驱动的智能体中, 对于请求 1 和请求 2 的文本, 当相似性阈值为 0.75 时, 由于 2 个请求中关键参数不同, 经测试, 直接对请求原始文本进行相似性界定会将 2 个请求判断为不相似, 而事实上请求 2 是可以复用大模型为请求 1 生成的计划. 因此, 需要设计新的方法来界定大模型驱动的智能体中的请求文本之间的相似性.

挑战 2: 如何识别关键参数. 在界定了请求文本间的相似性后, 若 2 个请求相似, 那么后到达的请求便可复用之前的请求所对应的计划. 在复用计划时, 需要替换关键参数. 在请求 1 和请求 2 中, 关键参数是时间(后天对应明天)、出发地(合肥对应长沙)、目的地(北京对应上海). 直觉上可以通过字符串匹配的方式, 在请求 2 和请求 1 对应的位置上使用相应参数进行替换. 但自然语言中存在同义词、改变语序等表达方式, 如请求 2 也可以表述为“给我定一下, 长沙去上海的票, 明天的”, 字符串匹配通常很难解决这类问题. 因此, 需要一个能感知语义的关键参数识别方法.

挑战 3: 如何正确、有效地复用计划. 当前, 智能体输出的计划通常为非结构化的文本. 即使已经识别到了关键参数, 也很难确定每个参数应该被应用到哪个子任务中, 难以正确、有效地复用计划. 因此, 需要设计一个能够将非结构化文本转换为结构化执行计划的方法, 以实现有效的复用计划.

为解决上述挑战, 本文提出 AgentReuse, 一个面向大模型驱动的智能体的计划复用机制, 并以毫秒级的额外延迟代价实现了 AgentReuse.

本文主要贡献有 3 个方面:

1)针对大模型驱动的智能体返回响应时间长的问题, 形式化描述并分析了大模型驱动的智能体的工作流程, 提出了对大模型驱动的智能体生成的计划进行复用的研究思路.

2)本文提出并以低额外代价实现了面向大模型驱动的智能体的计划复用机制 AgentReuse. AgentReuse 通过利用请求文本间语义的差异性进行关键参数识别, 将关键参数提取出来后, 基于意图分类方法, 利用文本间语义的相似性对用户请求文本进行相似性界定, 以实现计划复用.

3)为验证本文所提机制 AgentReuse 的有效性, 本文使用真实世界的数据集进行实验. 实验结果表明, 与现有方法相比, AgentReuse 实现了计划的有效复用, 有效复用率为 93%, 相似性界定的 $F1$ 分数提高了 6.8 个百分点, 准确率提高了 13.06 个百分点, 延迟降低了 60.61%.

1 背景知识与相关工作

针对本文的研究主题, 本节将对大模型驱动的智能体和缓存与复用方法这 2 方面的背景知识和近期研究进展进行简要介绍.

1.1 大模型驱动的智能体

智能体(agent)是一个具备环境感知、决策制定及动作执行能力的自主算法系统, 一直是学术界和工业界研究的重点. 研发智能体的初衷在于模拟人类或其他生物的智能行为, 旨在自动化地解决问题或执行任务. 然而, 传统智能体面临的主要挑战是它们通常依赖于启发式规则或受限于特定环境约束, 很大程度上限制了它们在开放和动态场景中的适应性与扩展性[17]. 由于大模型在解决复杂任务方面展现出了非常优秀的能力, 越来越多的研究工作开始探索将大模型作为智能体的控制中心以提高智能体在开放领域和动态环境中的性能[18], 如以 AutoGPT[19], AutoGen[12], HuggingGPT[20], MetaGPT[21] 为代表的一系列大模型驱动的智能体.

针对大模型驱动的智能体, 研究人员提出, 智能体=大模型+记忆+规划+工具使用, 即以大模型为核心控制器, 辅以规划、记忆、使用工具等能力, 通过协调不同能力, 完成用户提交给智能体的任务[9]. 文献 [21] 设计了 MetaGPT, 为多个智能体进行分工, 共同完成计算机软件的开发与调试. 文献 [22] 提出了 ChemCrow, 整合了 17 种由专家设计的工具, 增强了大模型在化学领域的能力. 文献 [23] 提出了一种大模型驱动的选址推荐引擎, 综合利用多种地理编码工具和选址模型实现选址任务的智能规划、执行和

归因，提升了大模型在地理空间任务上的表现. 文献 [24] 提出了 ResearchAgent, 模仿人类撰写科研论文的步骤，引导大模型生成包括问题识别、方法开发和实验设计等在内的完整研究思路. 本文针对典型大模型驱动的智能物联网应用，即个人智能体展开研究. 对于个人智能体，VIVO 发布了基于蓝心大模型（BlueLM）[25] 的蓝心小 V，小米公司的小爱同学也接入了大模型[4]，谷歌公司发布的 Pixel 8 Pro[26] 和 Bard 助理[27] 均集成了大模型，微软公司将大模型能力集成到了 Microsoft Copilot[28] 中并推出了 Copilot+PC[29]. 此外，在学术界，文献 [30] 提出的 AppAgent 和文献 [31] 提出的 AutoDroid，可以通过自主学习和模仿人类的点击和滑动手势，在手机上执行各种任务. 本文主要面向个人智能体展开研究，通过分析用户提交到智能体的任务请求文本间的相似性，对大模型生成的计划进行复用，降低智能体响应延迟.

1.2 缓存与复用方法

在计算机体系结构中，缓存旨在弥合内存访问延迟和处理器处理速度之间的差距[32-34]. 如今，缓存的概念已经扩展到不同领域，并在提高计算机系统性能中发挥了重要作用. 位于不同类型设备之间、用于消除访问时间差异带来的影响的结构可以被称为缓存，例如磁盘缓存[35-37]、CDN 缓存[38-40]、Web 缓存[41-43] 和容器缓存[44-46] 等. 缓存的有效性来源于请求的时间局部性，即在计算机应用中，最近刚刚被请求过的内容容易再次被请求[47]. 把刚被请求的内容缓存起来，供下次请求时复用，能有效降低响应延迟.

近 2 年，研究人员探索了缓存与复用技术在大模型中的应用. 研究人员针对大模型中的 KV Cache 展开了研究，基于 Self-Attention 计算特性，文献 [48] 提出 KV Cache 管理方法 Paged Attention; 文献 [49] 基于 token 的重要性不同提出 H2O 对 KV Cache 进行压缩; 文献 [50] 针对大模型处理长上下文时面对的系统层面的挑战，提出 CacheGen 对 KVCache 进行动态重新计算. 文献 [14−16] 基于请求文本间的语义相似性对大模型产生的响应进行了缓存和复用. 文献 [14] 提出的基于联邦学习的缓存与复用方法考虑了用户隐私和相似性检索的代价问题，文献 [15] 对大模型响应的缓存与复用方法进行了理论分析，文献 [16] 提供了开源工具以便开发者为大模型加入缓存与复用机制. 对于文生图的扩散模型，文献 [51] 提出的 NIRVANA 方法，通过重用先前图像生成期间创建的中间噪声状态来减少迭代去噪步骤.

本文主要针对大模型驱动的智能体产生的计划进行缓存和复用，现有方法提出的均是针对大模型响应的缓存与复用方法，无法直接应用到大模型驱动的智能体中.

2 大模型驱动的智能体工作流程与研究目标

本节将形式化描述本文所研究的大模型驱动的智能体的工作流程并简要介绍研究目标.

如图 1 所示，当本文研究的大模型驱动的智能体面向用户进行服务时，①智能体首先接收用户提交的任务请求 q_i，q_i 为自然语言文本. ②接收到请求后，智能体使用大模型的规划能力，生成针对 q_i 的计划 p_i. ③得到计划 p_i 后，智能体调用搜索引擎、代码解释器等各类工具，执行计划 p_i，得到响应 r_i，r_i 可以是用户想要得到的自然语言文本，也可以是任务执行后返回的响应代码（如 HTTP 服务中的 200 代码）. ④智能体将响应返回给用户.

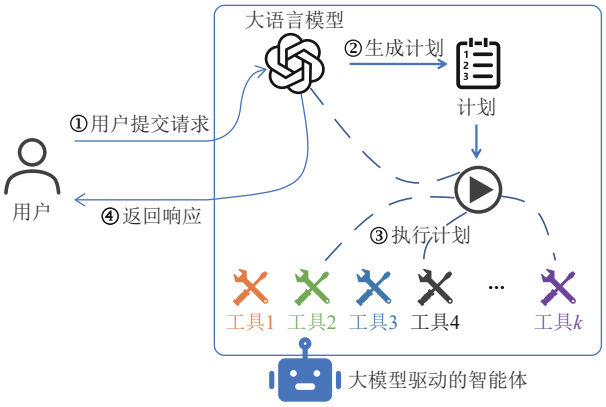

Fig. 1 Workflow of LLM-driven agent

图 1 大模型驱动的智能体工作流程

不失一般性的，我们定义 $Q=\{q_1,q_2,\cdots\}$，$P=\{p_1,p_2,\cdots\}$，$R=\{r_1,r_2,\cdots\}$. 对于请求 q_i，将生成计划文本的时间称为计划生成延迟 t_{p}^i，具体来说，t_{p}^i 指的是从智能体接收到 q_i 这一时刻至计划文本 p_i 生成结束（即生成结束符<EOS>）的时刻之间的时间间隔. 将执行计划的时间称为执行延迟 t_{e}^i，指的是计划文本 p_i 生成结束的时刻至产生响应 r_i 的时刻之间的时间间隔. $t_{\mathrm{t}}^i=t_{\mathrm{p}}^i+t_{\mathrm{e}}^i$ 即为对于请求 q_i 的响应延迟.

如前文所述，在智能体接收到的任务请求中，约有 30% 的请求在语义上是相同的或相似的，如果能够将相同或相似的请求识别出来，复用智能体针对先前请求生成的计划，就可以省去计划生成延迟 t_{p}^i，减少响应延迟，优化用户使用智能体的体验. 在本文中，我们将事实上可以复用先前计划的请求称为可

复用请求,将事实上不可复用先前计划的请求称为不可复用请求.

在对请求间的相似性进行界定时,如果不可复用请求被判定为可复用请求,那么将复用错误的计划,进而返回给用户错误的响应;若将可复用请求判定为不可复用请求,那么将带来额外的计划生成延迟.因此,设计一个能准确界定请求间相似性的方法十分必要.此外,对于可复用请求,当复用先前生成的计划时,需要将原计划中的部分参数替换为当前请求中的相关参数,目前尚无方法能实现参数的有效替换,即不能有效地完成计划复用.因此,我们还需要设计一个能有效复用计划的方法.

本节主要介绍了大模型驱动的智能体的工作流程,主要包括接收用户请求、生成计划、执行计划和返回响应4个部分,并对其进行了形式化描述.此外,本节定义了可复用请求、不可复用请求,明确了设计能准确进行相似性界定和有效复用计划的方法的重要性.

3 复用机制设计

本节将介绍我们提出的面向大模型驱动的智能体的复用机制AgentReuse的具体内容,并使用例子解释AgentReuse的工作流程.

在大模型驱动的智能体为用户提供服务时,接收请求q_i,利用大模型的规划能力得到计划文本p_i,最后利用工具执行计划得到响应r_i.其中,计划是由大模型生成的,计划生成延迟t_p与计划文本的长度成正比,计划文本越长,t_p越大.使用AutoGen作为智能体框架,OpenAI的GPT-4 API作为大模型,在真实世界的数据集[13]上进行100次测试,结果表明,平均计划生成延迟为31.8 s.之前的研究[14]和我们在真实世界数据集上的测试均表明,约有30%的请求是相同或相似的,对于相同或相似的请求,如果能够复用先前请求的计划或响应,就能够在一定程度上减少响应延迟.基于此,我们的目标是面向大模型驱动的智能体设计一个计划复用机制,以达到降低响应延迟的目的.

如算法1和图2所示是本文提出的复用机制AgentReuse,该机制通过利用请求文本间语义的相似性和差异性,对大模型驱动的智能体的计划进行有效复用.

算法1. AgentReuse.

输入:请求$Q=\{q_1,q_2,\cdots\}$,缓存,意图类别$\{c_0,c_1,\cdots,c_n\}$,相似性阈值γ;

输出:响应$R=\{r_1,r_2,\cdots\}$.

① 初始化缓存为空;
② for each q_i
③ 对q_i进行向量化得到$\boldsymbol{E}(q_i)$;
④ 对$\boldsymbol{E}(q_i)$进行意图分类和参数识别,分类结果为c_i,识别出的参数为$para_i$;
⑤ if $c_i \neq c_n$ /*c_n为未定义意图*/
⑥ 从q_i中提取出参数$para_i$,得到q_i^-和$\boldsymbol{E}(q_i^-)$;
⑦ 在缓存中对$\boldsymbol{E}(q_i^-)$进行相似性搜索;
/*在同一个c_i内进行*/
⑧ if 最高的相似性分数大于等于γ
⑨ 复用q_m的计划p_m; /*缓存中$\boldsymbol{E}(q_m^-)$与$\boldsymbol{E}(q_i^-)$相似度最高且大于等于γ*/
⑩ 利用参数$para_i$,使用工具执行计划p_m得到响应r_i;
⑪ else
⑫ 调用大模型生成计划p_i;
⑬ 使用工具执行计划p_i得到响应r_i;
⑭ 将$\langle\boldsymbol{E}(q_i^-),p_i,c_i\rangle$存入缓存;
⑮ end if
⑯ else
⑰ 调用大模型生成计划p_i;
⑱ 使用工具执行计划p_i得到响应r_i;
⑲ end if
⑳ end for
㉑ return 响应$R=\{r_1,r_2,\cdots\}$.

接下来,通过例子q_i="预定后天合肥到北京的票"来解释AgentReuse.首先,我们将请求文本q_i进行向量化并对其进行意图分类.意图分类是指根据文本的语义将请求文本分类到不同的类别中,即利用不同文本间的语义相似之处,将在意图上相似的文

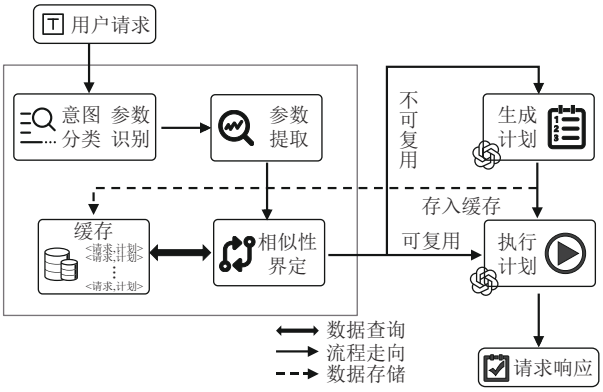

Fig. 2 Flowchart of AgentReuse

图2 AgentReuse流程图

本分类到相同的意图类别之中. 在个人助手中, 意图类别主要有查询、播放、发送、下载、预定等粗糙的分类, 当前例子q_i的意图为“预定”. 此外, 槽位填充任务与意图分类任务通常联合执行[52], 使用槽位填充来识别关键参数并将参数“(后天, 合肥, 北京)”暂存到$para_i$中. 之后, 在意图类别“预定”内, 进行关键参数提取和相似性界定. 对于参数提取, 在例子中, 需要被提取出来的参数为“(后天, 合肥, 北京)”, 在请求文本q_i中删去参数, 得到q_i^-=“预定到的票”. 这样, 便能在“预定”类内与预定餐厅、预定会议室等任务区别开, 找到与“预定到的票”更相似的请求, 并复用相似的请求计划. 其中, 相似与否由相似性阈值γ界定, γ是一个经验值, 本文设置γ的默认值为 0.75. 为了有效地复用计划, 在进行计划复用时, 使用之前识别出的参数$para_i$作为输入, 即利用请求文本语义间的差异性. 当缓存中没有与q_i^-相似的请求时, 则调用大模型生成计划然后使用工具执行计划得到响应r_i, 并将$\langle \boldsymbol{E}(q_i^-), p_i, c_i\rangle$存入缓存, 其中$\boldsymbol{E}(q_i^-)$是向量化后的$q_i^-$, p_i是大模型生成的计划, c_i是文本q_i的意图类别. 当q_i不在“预定”的意图类别内, 即意图类别为c_n时, 则直接调用大模型生成计划p_i并执行计划得到响应r_i. 值得注意的是, 在图 2 中, 参数提取、相似性界定、缓存查找是在当前意图类别内部进行的, 这样可以降低搜索成本.

在 AgentReuse 中, 本文通过利用请求文本间语义的相似性对请求文本进行意图分类, 同时利用请求文本间语义的差异性来识别和提取请求文本中的关键参数. 我们利用识别到的关键参数对请求文本进行相似性评估和参数替换, 以实现对大模型驱动的智能体接收到的请求进行准确的相似性界定和计划的有效复用. 此外, 由于用户提交的请求可能会涉及到实时变化的信息, 如机票信息, 因此, 本文对计划进行复用, 而不是对响应进行复用.

相比于现有大模型驱动的智能体的执行过程, AgentReuse 在存储和延迟上带来了额外代价.

1)存储代价. AgentReuse 需要使用模型对文本进行向量化以及对文本进行意图分类, 模型在运行时, 会占用显存. 向量化后的文本以及大模型产生的计划需要被缓存起来, 占用存储空间.

2)延迟代价. 相比于现有的执行过程, AgentReuse 引入的新流程均会带来额外的运行延迟, 其中延迟较大的部分是使用模型对用户请求进行意图分类和参数识别, 以及将用户的请求与已缓存的请求进行相似性界定.

值得注意的是, 现有的个人助手需要使用模型对用户请求进行意图分类[53-54]. 因此, 意图分类和参数识别带来的额外存储、延迟代价可不计入 AgentReuse 的代价中. 对于额外代价的具体分析和讨论将在第 5 节中进行.

本节介绍了本文的核心内容, 面向大模型驱动的智能体的计划复用机制 AgentReuse. 为了对请求的相似性进行界定, AgentReuse 对请求进行意图分类和关键参数识别, 将关键参数提取后再进行相似性界定, 以提高相似性界定的性能, 若缓存中存在能被复用的计划, 则复用、执行该计划; 若不存在能被复用的计划, 则生成一个新的计划, 并把新的计划存储到缓存中. 本节最后还对 AgentReuse 带来的额外代价进行了简要分析. 为了对计划进行有效复用, 本文采用增加提示词的方法以获得结构化的计划, 将在第 4 节进行介绍.

4 复用机制的实现

基于前文提出的面向大模型驱动的智能体的计划复用机制 AgentReuse, 本节旨在介绍实现 AgentReuse 的相关技术. 值得注意的是, 本节采用的相关技术只是实现 AgentReuse 的一种方法, 对于不同的性能需求、额外代价限制、输入文本语言类型, 可以选取不同的技术路线.

对于意图分类和参数识别, 本文采用 bert-base-chinese 模型[55], 该模型是 BERT 模型的中文版本, 具有强大的语言理解能力. 模型采用双向 Transformer 架构, 预训练大量中文语料数据, 在多个中文自然语言处理任务上表现出色, 如文本分类、命名实体识别和情感分析等, 为中文自然语言处理任务提供了坚实的基础.

对于文本向量化, 本文采用 m3e-small 模型[56], m3e 指的是 Moka Massive Mixed Embedding, 具体意义是: 此模型由 MokaAI 训练, 通过千万级 (massive) 的中文句对数据集进行训练, 支持中英双语(mixed)的同质文本相似度计算, 是文本向量化(embedding)模型, 可以将自然语言转换成稠密的向量. m3e-small 参数的数量为 2 400 万, 产生的向量维度为 512.

对于文本语义相似度的计算, 本文采用余弦相似度. 向量可以捕获单词或句子的语义含义, 具有相似含义的单词或句子将具有相似的向量表示. 例如, “火车”和“高铁”这 2 个词虽然文本上不同, 但在语义上是相似的. 因此, 在将它们向量化后, 这 2 个词

语对应的向量在向量空间中是接近的. 余弦相似度通常用于量化向量之间的相似度, 通过取 2 个向量之间的角度的余弦来计算. 余弦相似度的范围为−1~1, 其中−1 表示 2 个向量方向完全相反, 0 表示 2 个向量正交, 1 表示 2 个向量的方向相同.通常, 在文本处理中, 余弦相似度值越接近 1, 两个文本就越相似. 对于向量 $\boldsymbol{V}_1$ 和 $\boldsymbol{V}_2$, 余弦相似度为

$$\cos\theta = \frac{\boldsymbol{V}_1 \cdot \boldsymbol{V}_2}{\|\boldsymbol{V}_1\|\|\boldsymbol{V}_2\|} \tag{1}$$

对于向量相似性搜索, 在本文的技术实现中, 采用 Facebook AI Similarity Search(faiss)[57] 中的 indexFlatIP 索引进行高效相似性搜索. faiss 是 Facebook 开源的相似性搜索库, 为向量提供高效相似性搜索和聚类, 支持十亿级别向量的搜索, 是目前最为成熟的近似近邻搜索库. 值得注意的是, faiss 中没有直接提供余弦相似度计算, 本文选用的 indexFlatIP 进行了点积计算, 在进行余弦相似度计算时, 还需要先对向量进行归一化.

此外, 由于 AutoGen 等大模型驱动的智能体生成的计划通常是自然语言文本, 为了对计划进行存储和有效复用, 本文对自然语言文本的计划进行了结构化处理. 在 AutoGen 使用大模型生成计划提示词中, 我们提示大模型在生成计划时对子任务间的依赖信息进行简单标注, 基于依赖标注, 对计划文本进行处理便可得到结构化后的计划. 此外, 在 AutoGen 等智能体框架中, 为了系统安全性的考虑, 对于工具的使用通常在容器中进行[58-60], 因此, 在步骤中需要记录使用的容器镜像信息, 以便后续复用. 对于请求 q_i 的自然语言文本计划 p_i, 将计划 q_i 表示为:

步骤 1, 步骤描述, 容器镜像, 输入, 依赖, 输出;

步骤 2, 步骤描述, 容器镜像, 输入, 依赖, 输出;

⋮

步骤 k, 步骤描述, 容器镜像, 输入, 依赖, 输出.

以“预定后天合肥到北京的票”产生的计划为例, 本文将其转换为如下结构:

步骤 1, 查询日期, query_date, 日期描述, 无依赖, 日期;

步骤 2, 查询天气, query_weather, (地点, 日期), 依赖步骤 1, 天气;

步骤 3, 查询机票, query_flight, (出发地, 目的地, 日期), 依赖步骤 1, 机票信息;

步骤 4, 查询火车票, query_train, (出发地, 目的地, 日期), 依赖步骤 1, 火车票信息;

步骤 5, 价格比较, compare_price, (机票信息, 火车票信息), 依赖步骤 3、4, 比较结果;

步骤 6, 查询日程, query_schedule, 日期, 依赖步骤 1, 个人日程;

步骤 7, 结合前述信息和个人偏好推荐出行方案, call_prefer, 前述所有信息, 依赖步骤 2、5、6, 推荐结果.

图 3 展示了上述计划的执行图, 对于其中每一个步骤, 只有其依赖的步骤执行完成后才能执行当前步骤. 其中步骤 2、3、4、6 依赖步骤 1; 步骤 5 依赖步骤 3、4; 步骤 7 依赖步骤 2、5、6. 在实际执行中实现各个步骤的方式如下: 使用开放式的 API 完成步骤 1、步骤 2, 使用开放式 API 或携程、12306 等应用程序完成步骤 3 和步骤 4, 使用 Python 代码完成步骤 5, 使用日历应用程序完成步骤 6, 最后通过调用大模型完成步骤 7.

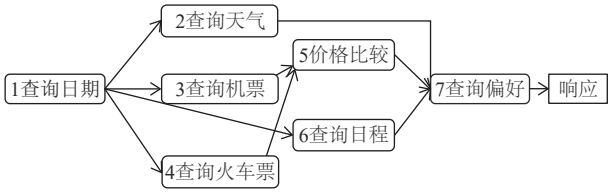

Fig. 3 Execution graph for the plan generated by the request

图 3 请求生成的计划的执行图

本节介绍了本文实现 AgentReuse 时的所采用的技术方法, 包括使用采用 bert-base-chinese 作为意图识别模型, 使用 m3e-small 作为文本向量化模型以及采用余弦相似度来评估文本语义相似度等内容. 正如本节最开始提到的, 本节所介绍的只是实现 AgentReuse 的方案之一, 对于不同语言、不同的性能需求、不同的资源占用限制, 可以选用不同的模型和相关技术.

5 实验与结果分析

为了评估所提出机制 AgentReuse 的有效性和额外代价, 本文基于真实世界数据集, 进行了一系列实验. 本文首先对实验设置进行详细说明, 然后介绍评估指标, 最后展示实验结果并对额外代价进行分析. 实验结果表明, 相比于现有方法, AgentReuse 实现了对计划的有效复用, 计划有效复用率为 93%, 相似性界定方法的 $F1$ 分数比现有方法高 6.8 个百分点, 准确率高 13.06 个百分点, 延迟降低 60.61 个百分点. 相比于现有大模型驱动的个人智能体的执行过程, AgentReuse 的额外显存占用约 100 MB, 处理每个请求的内存占用不超过 1 MB, 延迟不超过 10 ms.

5.1 实验设置

本文实验使用SMP数据集[61]，其中包括2 664条大模型驱动的个人智能体的任务请求，如：“明天早上到成都的机票都有哪些时间”“来一首李清照的词”等文本请求，包括“LAUNCH”“QUERY”“ROUTE”等23种意图类别.

本文实验在如下配置的服务器上进行. 操作系统为Ubuntu 20.04.6 LTS, CPU为Intel® Xeon® Gold 6330 CPU @ 2.00 GHz, GPU为NVIDIA A100 80 GB, Python版本为3.10, CUDA版本为11.4, 服务器内存大小为503 GB.

为了测试AgentReuse的有效性，本文选取2种其他研究人员提出的方法作为对比方法：

1）GPTCache[16]. 用于对大模型响应进行缓存与复用的机制. 在进行相似性界定时，直接将请求原始文本进行向量化，计算语义相似度.

2）MeanCache[14]. 用于对大模型响应进行缓存与复用的机制. 对请求文本进行主成分分析，降维到64维后再进行相似性界定.

此外，为了验证AgentReuse中不同模块的有效性，即进行消融实验，本文还对2种方法进行了测试：

1）OneIntent.在本文实现的AgentReuse中进行相似度界定时，对所有已经缓存的内容进行检索，而不是仅在当前意图类别内进行检索.

2）WithArgs.在本文实现的AgentReuse中进行相似度界定时，不进行参数提取，直接使用向量化后的请求原始文本进行相似性搜索.

5.2 评估指标

需要对AgentReuse相似性界定的性能、复用计划的有效性、额外代价和性能增益进行评估，本节将分别介绍用于评估上述实验效果的评估指标.

对于相似性界定，本文主要使用$F1$分数和准确率来评估不同方法的性能. 在传统的缓存与复用系统中，性能通过缓存命中率和缓存未命中率来衡量. 缓存命中意味着数据或查询请求从缓存中检索到了，而缓存未命中则表示缓存中没有此数据或请求. 针对自然语言文本的语义相似性界定，引入了更细致的分类：真正例（true positive, TP）、假正例（false positive, FP）、真负例（true negative, TN）和假负例（false negative , FN）.

真正例：可复用请求被正确识别为可以复用. 假正例：不可复用请求被错误识别为可复用，实际上并不能复用. 真负例：不可复用请求被正确识别为不能复用. 假负例：可复用请求被错误识别为不可复用，实际上是可以复用的.

本文基于上述分类采用精确率、召回率、F_β分数和准确率这4个评估指标，对不同评估指标，其意义和计算公式为：

1）精度（*Precision*）. 用来表示被识别为可复用的请求中可复用的比例.

$$Precision = \frac{TP}{TP+FP}. \tag{2}$$

2）召回率（*Recall*）. 用来表示被正确识别为可复用的请求占实际上应该可以复用的请求的比例，也可以称之为查全率.

$$Recall = \frac{TP}{TP+FN}. \tag{3}$$

3）F_β分数（F_β Score）. F_β分数是精度和召回率的加权调和平均，根据β值平衡两者. 当$\beta > 1$时，对召回率赋予更多权重；当$\beta<1$则强调精度. 本文使用$F1$分数作为评估指标之一.

$$F_\beta\text{分数} = (1+\beta^2)\cdot\frac{Precision\times Recall}{\beta^2\times Precision + Recall}. \tag{4}$$

4）准确率（*Accuracy*）. 用来表示正确识别的查询与所有情况的比值.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}. \tag{5}$$

在上述4个指标中，精度用来表示被识别为可复用的请求中有多少是真正可以复用的；召回率用来表示被正确识别为可复用的请求占实际上应该可以被复用的请求的比例；F_β分数是精度和召回率的加权调和平均；准确率用来表示被正确识别的请求的比例. 在本文的实验中，F_β分数（本文采用$F1$分数）是能更加综合地体现方法性能的评估指标.

本文使用复用有效率作为评估复用计划的有效性的量化评估指标，详见5.3节.

对于使用AgentReuse的额外代价，本文采用处理每个请求的额外延迟和占用的额外内存（包括显存）作为评估指标.

对于性能增益的评估，由于使用不同复用机制对于执行计划的延迟没有影响，因此，在本节中进行性能增益分析时，只考虑计划生成延迟和使用机制的额外代价，且假设所有的非真正例均需要进行一次计划生成.

5.3 结果分析

本节主要用于介绍相似性界定、计划复用有效性的实验结果并对实验结果进行分析.

首先，分析相似性界定的实验结果，只有在相似性界定结果准确的情况下进行计划的复用才会有效，

否则会造成可复用请求并没有被判定为可复用, 不可复用请求被判定为可复用先前计划进而产生错误响应等后果.

表 1 展示的是在默认设置情况下, 即相似性阈值 γ=0.75 时的相似性界定实验结果. 本文所提机制 AgentReuse 的 $F1$ 分数和准确率最高, 分别为 0.971 8 和 0.945 9. 在对比方法中, AgentReuse 的变种, 相似性检索时在全部已缓存内容中进行检索而不是仅在当前意图类别内进行相似性检索的 OneIntent 方法效果与 AgentReuse 十分接近, 但其运行延迟比 AgentReuse 慢 17.66 个百分点, 对于延迟额外代价的讨论详见后文. 相比于针对大模型响应复用机制的 GPTCache, AgentReuse 将 $F1$ 分数和准确率分别提高了 6.8 个百分点和 13.06 个百分点.

接下来, 我们调整相似性阈值 γ, 分别设置阈值 γ =0.75, 0.80, 0.85, 0.90, 0.95.观察不同方法的性能表现变化. 图 4(a)~(d) 分别展示当阈值发生变化时, 5 种方法的 $F1$ 分数、精度、召回率和准确率的变化. 对于 $F1$ 分数和准确率, AgentReuse 及其变种 OneIntent 的分数较高. 整体上, 当阈值变大时, 召回率和准确率在逐渐下降, 精度在逐渐上升, 这是因为当阈值变大时, 复用机制变得更加保守, 只有相似度极高的请求才会被判定为可以复用, 导致原本可进行复用的请求被判定为不可复用, 召回率下降较多.

Table 1 Experimental Results for Similarity Evaluation

表 1 相似性界定实验结果

方法/指标	$F1$ 分数	精度	召回率	准确率
GPTCache	0.910 0	0.988 7	0.842 8	0.836 6
MeanCache	0.621 1	0.999 1	0.450 6	0.461 4
OneIntent	0.971 2	0.985 7	0.957 1	0.944 4
WithArgs	0.903 5	0.996 7	0.826 2	0.827 2
AgentReuse	0.971 8	0.993 1	0.951 3	0.945 9

此外, 通过表 1 和图 4 我们发现, 作为 AgentReuse 的变种, WithArgs 的性能较差, WithArgs 在进行相似性界定时, 没有对原请求进行关键参数提取. 图 5 展示了当相似性阈值 γ=0.75 时, AgentReuse 和 WithArgs 对于同一个请求的相似性判定的详细情况, 其中, “应复用”表示事实上该请求可以复用之前的请求生成的计划, “不应复用”表示在事实上该请求无法复用之前的请求生成的计划. 为了展示得更加清楚, 仅选取了请求 ID 从 100 到 500 的结果. 从图 5 中可以看到, 对于同一个请求, 由于 AgentReuse 进行了参数提取, 即去掉了相似句子中的不相似部分

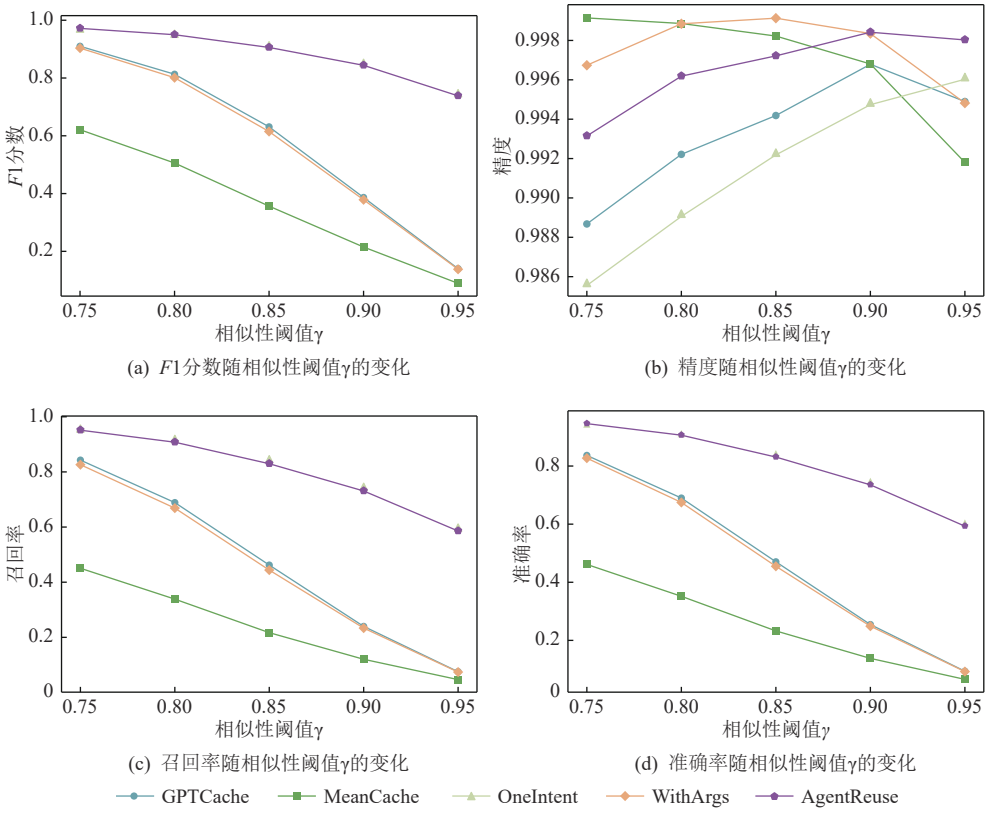

Fig. 4 Effect of variation in similarity threshold on the performance of different methods

图 4 相似性阈值变化对不同方法性能的影响

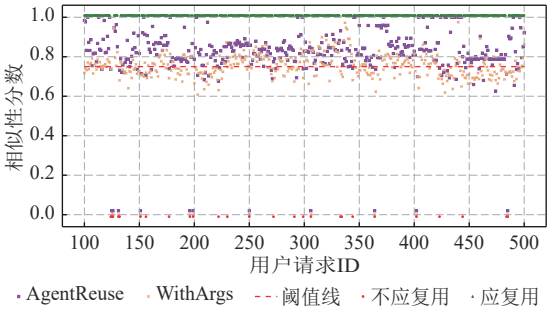

Fig. 5 Comparison of similarity scores calculated by AgentReuse and WithArgs.

图 5 AgentReuse 与 WithArgs 计算得到的相似性分数的比较

(即关键参数), AgentReuse 得到的相似性分数更高, 从而达到了更高的召回率(0.951 3 与 0.826 2).

为了分析关键参数提取对结构化的计划复用是否有效, 我们使用 AutoGen 作为智能体框架进行了实验. 实验测试标准为对于同一用户请求, 执行 2 次测试: 第 1 次测试使用 AgentReuse, 对用户请求进行关键参数提取, 复用已存储在缓存中的结构化计划, 执行计划并得到响应; 第 2 次测试则直接将请求提交给 AutoGen, 生成并执行计划, 最后得到响应. 比较 2 次测试的响应, 若第 1 次测试结果与第 2 次测试结果相同, 则说明参数提取和计划复用有效. 我们对 20 个请求进行了实验, 针对每个请求的实验重复 5 次, 即共 100 次实验, 其中 93 次实验中 2 次测试的响应相同, 即复用有效率为 93%. 20 个请求中含有 4 个预定类请求、4 个启动程序类请求、2 个查询类请求、2 个翻译类请求、2 个创建类请求、2 个搜索类请求、2 个下载类请求、1 个导航类请求以及 1 个非定义请求.

本节对实验的结果进行了分析. 实验结果表明, 在使用 AgentReuse 机制进行相似性界定时(相似性阈值 γ=0.75), $F1$ 分数可达 0.971 8, 准确率可达 0.945 9, 均高于对比方法, 此外, 我们分析了当相似性阈值发生变化时不同方法的评估指标的变化, 以及去除请求中关键参数的有效性的分析. 最后, 我们对复用的有效性进行了分析.

5.4 额外代价分析

相比于现有的大模型驱动的智能体执行过程, 本文设计并实现的 AgentReuse 使用了额外的模型, 进行缓存时占用了额外的存储空间, 也相应地引入了额外的延迟. 本节旨在通过实验测试表明本文实现的 AgentReuse 引入的额外代价是可以接受的.

5.4.1 存储额外代价分析

1)模型占用显存大小. 对于本文使用的用于意图分类和识别参数的 bert-base-chinese 模型, 参数量约为 1.023 亿, 每个参数占用 4B, 理论上占用显存约 390.24 MB, 经实验测试, 系统分配预留显存为 444.00 MB, 模型实际使用显存为 391.16 MB. 对于本文使用的文本向量化模型 m3e-small, 参数量为 2 400 万, 理论上占用显存约 91.55 MB, 经测试, 系统分配预留显存为 102.00 MB, 模型实际使用显存为 92.12 MB. 此外, 在模型执行过程中还会带来额外的显存开销, 但总体显存占用依旧为 MB 级别, MB 级别的额外代价对于现今的智能手机等设备来说是可以接受的.

2)向量化后的文本占用. 通常, 向量化后的文本是以 32 b 浮点数存储的, 每个浮点数占用 4 B 内存, 本文使用的 m3e-small 会将文本向量化为 512 个维度, 因此, 对于每个向量化后的文本, 内存占用为 2 KB. 除了向量化后的文本本身之外, 为了进行相似度查找, 还需要存储意图类别, 使用的 faiss 索引也需要占用额外内存, 经测试, 内存占用不超过 1 MB. 总的来说, 每添加一个 512 维的向量化后的文本, 最基本的额外内存需求是 2 KB 加上意图类别数据和索引库的开销, 对于每个请求, 额外存储代价通常不会超过 1 MB.

3)计划文本占用. 本文实现的 AgentReuse 机制对计划文本进行了格式化存储, 即对于计划中的每个步骤, 格式化并存储为:

①步骤 k, 步骤描述, 容器镜像, 输入, 依赖, 输出.

②存储时采用 UTF-8 编码, 中文占用 3~4B, 数字、英文字母占用 1B,

③对于第 4 节中所描述的计划, 经测试, 总字符数为 303, UTF-8 编码后占用总字节数为 705, 平均每个步骤占用 100.71B. 总的来说, 对于每个计划文本, 存储带来的额外开销一般不超过 1 KB.

5.4.2 延迟额外代价分析

在大模型驱动的智能体的复用机制中, 对于 MeanCache, OneIntent, WithArgs, AgentReuse 四种方法, 其处理每个请求的额外延迟代价由意图分类、相似性搜索和其他延迟组成, GPTCache 没有对意图进行分类, 延迟由相似性搜索、其他延迟组成. 其他延迟主要包括文本向量化、将内容存储到缓存中等. 经 50 次测试后, 5 种方法处理每个请求的延迟代价的组成如图 6 所示.

图 6 表明, GPTCache 处理每个请求的总延迟最低, 仅 11.528 ms, 因为其并没有使用意图分类模型, 但如 5.3 节所述, 其相似性界定性能也较差. 在使用了意图分类的方法中, AgentReuse 处理每个请求的总

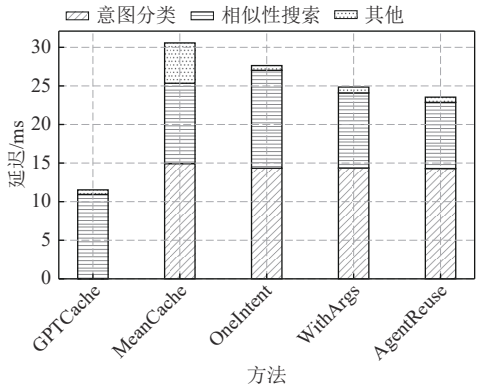

Fig. 6 Composition of latency of different methods to process each request

图 6 不同方法处理每个请求的延迟组成

延迟最低, 为 23.489 ms. 可以看到, AgentReuse 与 OneIntent 在相似性界定方面的 $F1$ 分数、准确率等性能接近, 但延迟代价更低. 对于 OneIntent(AgentReuse 的变种), 其延迟比 AgentReuse 大 17.66%, 二者的意图分类的延迟差距不大, 但 OneIntent 进行相似性搜索的延迟比 AgentReuse 大 47.94%, 这是因为在进行相似性搜索时, OneIntent 并没有在当前意图分类内进行搜索, 而是在所有已缓存的内容中进行搜索, 这增大了搜索空间, 搜索延迟自然变大. 此外, 由于 OneIntent 方法使用经参数提取后的请求文本在所有已缓存的内容中进行搜索, 会将部分不可复用的请求识别为可复用, 导致精度有所下降.

此外, 我们可以观察到, 除 MeanCache 外的 4 种方法的其他延迟均不超过 1 ms, 均为 5.228 ms. 这是因为 MeanCache 使用了主成分分析模型, 在对每个请求进行处理时, 均需要对请求文本进行主成分分析, 产生了额外的延迟代价. 对于 AgentReuse, 本文测量了 50 次格式化计划的延迟和将参数带入到可复用计划中的延迟, 二者平均延迟均小于 1 ms, 在后续中对此忽略不计.

综上所述, AgentReuse 机制带来的主要额外代价是显存占用约 500 MB, 存储每个向量化后文本和结构化计划的内存占用不超过 1 MB, 处理每个请求的延迟为 23.489 ms. 若考虑现有个人助手中已有意图分类模型, 那么 AgentReuse 机制将为显存占用带来约 100 MB 的额外代价, 处理每个请求的延迟约 9.206 ms. 对智能体产生的计划进行复用, 不仅可以节省计划生成延迟, 还能减少运行大模型的显存占用, 当前大模型的显存占用一般不少于 5 GB. 此外, 如果使用的是 OpenAI 等提供的付费 API, 还能够达到节省经费的目的. 因此, 使用 AgentReuse 带来的额外代价是可以接受的.

5.5 性能增益分析

本节对使用 AgentReuse 机制相比于不使用复用机制以及使用 GPTCache 方法的性能增益进行分析. 由于使用不同机制对于执行计划的延迟没有影响, 因此, 在本节中进行性能增益分析时, 只考虑计划生成延迟和使用机制的额外代价, 按照每次计划生成的延迟为 31.8 s, 且假设所有的非真正例均需要进行一次计划生成.

在本文对 AgentReuse 的实验中, 当相似性阈值 γ=0.75 时, 共有 2 464 个请求是真正例, 即本身请求是可复用请求, 且 AgentReuse 也将这些请求界定为可复用请求. 对于数据集中的 2 644 条数据, 如果全都需要调用大模型生成计划, 即不使用复用机制, 延迟为 84 079.2 s. 如果使用 AgentReuse, 按照处理每条请求需要 23.489 ms, 那么处理 2 644 条数据需要花费 62.1 s, 为 180 个非真正例请求生成计划的延迟是 5 724 s, 所以使用 AgentReuse 机制的延迟共 5 786.1 s. 对于 GPTCache, 共有 2 183 个请求是真正例, 其他 461 个请求是非真正例, 因此用 GPTCache 处理该数据集中的请求总延迟为 14 690.28 s. 所以, 相比于不使用复用机制, AgentReuse 可减少 93.12% 的延迟, 相比于面向大模型响应的缓存与复用方法 GPTCache, AgentReuse 可减少 60.61% 的延迟.

本节中, 我们使用 SMP 数据集进行实验, 并对实验结果进行了分析. 实验结果表明, AgentReuse 及其变体 OneIntent 的相似性界定效果优于其他现有方法. 对 AgentReuse 的额外代价分析表明, AgentReuse 相比于现有个人智能体的执行过程, 额外的显存占用约为 100 MB, 处理每个请求的额外内存占用不超过 1 MB, 处理延迟不超过 10 ms. 对性能增益进行分析表明, 在考虑了额外代价的情况下, AgentReuse 相比于不使用复用机制可减少 93.12% 的延迟, 相比于使用 GPTCache 可减少 60.61% 的延迟.

6 讨论与未来工作

6.1 端侧设备上的部署与优化

本文主要研究的是面向大模型驱动的智能体的计划复用机制, 基于对真实数据集的分析, 我们发现对大模型生成的计划进行缓存以供后续复用能够有效减少响应延迟, 基于此, 我们提出并以低额外代价实现了复用机制 AgentReuse. 本文工作的最终目标是在不依赖云端算力的智能手机上对 AgentReuse 进行

实验验证，然而，受现有大模型和大模型驱动的智能体的资源消耗的限制，本文临时在服务器上进行了实验. 令人期待的是，2024 年 4 月以来，苹果、微软等公司陆续发布了 OpenELM[62]、Phi-3-mini[63] 等端侧大模型，使得在个人笔记本、智能手机等端侧设备上部署和高效运行大模型和智能体成为可能. 另一方面，在进行端侧设备上的部署时，还需要对意图分类等模型进行优化. 由于受实验成本、GPT-4 API 请求速率以及智能体框架尚不完善等限制，本文仅针对部分实验数据进行了端到端实验，即从用户请求到返回响应的实验. 对于 API 请求成本，2024 年 5 月下旬，阿里云、科大讯飞和腾讯等厂商相继下调或免费开放了大模型 API，待智能体框架功能更完善、资源占用更少后，在端侧设备上进行更多的端到端实验将更加可行.

6.2 多意图分类与基于屏幕理解的智能体

本文的主要研究对象是大模型驱动的智能体，具体来说是大模型驱动的智能物联网应用，如小米公司的小爱同学、OPPO 公司的小布助手等. 传统上，一般用户只会对智能体产生 1 个输入，如“帮我预定后天合肥到北京的票”. 现今，随着社会的发展，智能汽车逐渐走入千家万户，智能汽车上的组件更多，此时，用户的 1 句话中可能包含多个任务，如“右前侧车窗降低，左后侧车窗抬高，打开雨刷器，播放陶喆的音乐”. 此时，需要采用多意图分类模型对文本中的意图进行分类和参数识别，分类后，将 1 句话分割为多个部分，再并行使用 AgentReuse 机制进行复用即可. 但当句子间有因果关系时，如“如果室内温度超过 28 摄氏度，就把空调开到制冷 26 摄氏度”，就不能简单地将句子分割为多个部分，需要引入新的模型对有因果关系的句子进行处理，经处理后再利用 AgentReuse 对之前生成的计划进行复用. 此外，AutoDroid[31] 和 MobileGPT[64] 等端侧智能体通过对屏幕 UI 结构的理解和操作完成用户的请求任务，如何对屏幕操作的执行进行缓存和复用，也是本文的后续研究内容之一.

7 结　论

在当今的人工智能时代，智能体被认为是通向通用人工智能的一条有前途的道路. 本文提出并实现该一个能有效复用大模型驱动的智能体所产生的计划的机制 AgentReuse，并在真实世界的数据集上测试该机制的有效性以及额外代价. 实验结果表明，相比于现有方法，AgentReuse 在请求间进行相似性界定时性能更好，通过对计划文本进行格式化存储，实现了对计划的有效复用，与不使用复用机制相比，能降低 93.12% 的延迟. 在实际真实运行环境中，智能体面临的任务往往更复杂，作为一个探索性的工作，我们设计并实现的 AgentReuse 可能无法处理所有的情况，需要结合更多实际场景中的特性来修改、完善. 如今，大模型已经融入了我们的生活，但大模型驱动的智能体还未得到大规模应用，我们希望本工作能够有助于推动智能体应用，尤其是大模型驱动的智能物联网应用的实际落地.

作者贡献声明：李国鹏提出问题解决思路、设计实验方案、撰写论文；吴瑞骐调研文献、参与问题解决方案讨论、完成实验、撰写论文；谈海生提出研究问题，参与问题解决方案讨论，对论文撰写与实验方案设计进行指导；陈国良对整体研究方向进行指导. 李国鹏与吴瑞骐具有同等贡献.

参　考　文　献

[1] Guangming Daily. Seize the opportunity, accelerate the development of new quality productive forces [EB/OL]. [2024-03-14]. https://news.gmw.cn/2024-03/14/content_37202598.htm (in Chinese)
(光明日报. 抢抓机遇，加快发展新质生产力[EB/OL]. [2024-03-14]. https://news.gmw.cn/2024-03/14/content_37202598.htm)

[2] Liu Yunhao. Introduction to Internet of Things[M]. Beijing: Sciences Press, 2017 (in Chinese)
(刘云浩. 物联网导论[M]. 北京：科学出版社，2017)

[3] Guo Bin, Liu Sicong, Liu Yan, et al. AIoT: The concept, architecture and key techniques[J]. Chinese Journal of Computers, 2023, 46(11): 2259−2278 (in Chinese)
(郭斌, 刘思聪, 刘琰, 等. 智能物联网：概念、体系架构与关键技术[J]. 计算机学报, 2023, 46(11)：2259−2278)

[4] Xiaomi Corporation. Xiaoai tongxue [EB/OL]. [2024-03-15]. https://xiaoai.mi.com/(in Chinese)
(小米集团. 小爱同学[EB/OL]. [2024-03-15]. https://xiaoai.mi.com/)

[5] Huawei Device Co., Ltd.. Wake up to intelligent voice [EB/OL]. [2024-03-15]. https://consumer.huawei.com/cn/emui-11/tips/smart-home-list/article5/ (in Chinese)
(华为终端有限公司. 智慧唤醒语音[EB/OL]. [2024-03-15]. https://consumer.huawei.com/cn/emui-11/tips/smart-home-list/article5/)

[6] Li Ge, Peng Xin, Wang Qianxiang, et al. Challenges from LLMs as a natural language based human-machine collaborative tool for software development and evolution[J]. Journal of Software, 2023, 34(10): 4601−4606 (in Chinese)
(李戈, 彭鑫, 王千祥, 等. 大模型：基于自然交互的人机协同软件开发与演化工具带来的挑战[J]. 软件学报, 2023, 34(10)：4601−4606)

[7] Dong Luna Xin, Moon Seungwhan, Xu Ethan Yifan, et al. Towards next-generation intelligent assistants leveraging LLM techniques[C]//Proc of the 29th ACM SIGKDD Conf on Knowledge Discovery and Data Mining. New York: ACM, 2023: 5792−5793

[8] Li Yuanchun, Wen Hao, Wang Weijun, et al. Personal LLM agents: Insights and survey about the capability, efficiency and security[J]. arXiv preprint, arXiv: 2401.05459, 2024

[9] Weng Lilian. LLM powered autonomous agents [EB/OL]. [2024-3-30]. https://lilianweng.github.io/posts/2023-06-23-agent

[10] Wang Lei, Ma Chen, Feng Xueyang, et al. A survey on large language model based autonomous agents[J]. Frontiers of Computer Science, 2024, 18(6): 1−26

[11] Xi Zhiheng, Chen Wenxiang, Guo Xin, et al. The rise and potential of large language model based agents: A survey[J]. arXiv preprint, arXiv: 2309.07864, 2023

[12] Wu Qingyun, Bansal Gagan, Zhang jieyu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation[J]. arXiv preprint, arXiv: 2308.08155, 2023

[13] Open-Assistant. Open assistant conversations dataset release 2 [EB/OL]. [2024-04-20]. https://huggingface.co/datasets/OpenAssistant/oasst2

[14] Gill W, Elidrisi M, Kalapatapu P, et al. Privacy-aware semantic cache for large language models[J]. arXiv preprint, arXiv: 2403.02694, 2024

[15] Zhu Banghua, Sheng Ying, Zheng Lianmin, et al. On optimal caching and model multiplexing for large model inference[C]//Advances in Neural Information Processing Systems. Cambridge, MA: MIT, 2023: 59062−59094

[16] Fu Bang, Feng Di. GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings[C]//Proc of the 3rd Workshop for Natural Language Processing Open Source Software. Stroudsburg, PA: ACL, 2023: 212−218

[17] Zhao Wayne Xin, Zhou Kun, Li Junyi, et al. A survey of large language models[J]. arXiv preprint, arXiv: 2303.18223, 2023

[18] Lin Chaofan, Han Zhenhua, Zhang Chengruidong, et al. Parrot: Efficient serving of LLM-based applications with semantic variable [C]//Proc of the 18th SENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2024: 929−945

[19] Mindstream. AutoGPT [EB/OL]. [2024-04-30]. https://autogpt.net/

[20] Shen Yongliang, Song Kaitao, Tan Xu, et al. Hugging GPT: Solving AI tasks with chatgpt and its friends in hugging face[C]//Advances in Neural Information Processing Systems. Cambridge, MA: MIT, 2023: 38154−38180

[21] Hong Sirui, Zhuge Mingchen, Chen Jonathan, et al. MetaGPT: Meta programming for a multi-agent collaborative framework[J]. arXiv preprint, arXiv: 2308.00352, 2023

[22] Bram M A, Cox S, Schilter O, et al. Augmenting large language models with chemistry tools[J]. Nature Machine Intelligence, 2024(6): 525−535

[23] Gao Yunfan, Yu Dongqing, Wang Siqi, et al. Large language model powered site selection recommender system[J]. Journal of Computer Research and Development, 2024, 61(7): 1681-1696(in Chinese)
(高云帆, 郁董卿, 王思琪, 等. 大语言模型驱动的选址推荐系统[J]. 计算机研究与发展, 2024, 61(7):1681-1696)

[24] Baek J, Jauhar K S, Cucerzan S, et al. ResearchAgent: Iterative research idea generation over scientific literature with large language models[J]. arXiv preprint, arXiv: 2404.07738, 2024

[25] VIVO. BlueLM-7B-Chat [EB/OL]. [2024-05-01]. https://huggingface.co/vivo-ai/BlueLM-7B-Chat

[26] Google.Googlepixel8pro[EB/OL]. [2024-05-21]. https://store.google.com/product/pixel_8_pro

[27] Hsiao S. Assistant with bard: A step toward a more personal assistant [EB/OL]. [2024-05-21]. https://blog.google/products/assistant/google-assistant-bard-generative-ai/

[28] Microsoft. Microsoft copilot [EB/OL]. [2024-05-01]. https://copilot.microsoft.com/

[29] Mehdi Y. Introducing Copilot+ PCs [EB/OL]. [2024-05-25]. https://blogs.microsoft.com/blog/2024/05/20/introducing-copilot-pcs/

[30] Zhang Chi, Yang Zhao, Liu Jiaxuan, et al. AppAgent: Multimodal agents as smartphone users[J]. arXiv preprint, arXiv: 2312.13771, 2023

[31] Wen Hao, Li Yuanchun, Liu Guohong, et al. AutoDroid: LLM-powered task automation in Android[C]//Proc of the 30th Annual Int Conf on Mobile Computing and Networking. New York: ACM, 2024: 543−557

[32] Wang Endong, Tang Shibin, Chen Jicheng, et al. Directory cache design for multi-core processor[J]. Journal of Computer Research and Development, 2015, 52(6): 1242−1253 (in Chinese)
(王恩东, 唐士斌, 陈继承, 等. 多核处理器目录缓存结构设计[J]. 计算机研究与发展, 2015, 52(6): 1242−1253)

[33] Sedaghati A, Hakimi M, Hojabr R, et al. X-cache: A modular architecture for domain-specific caches[C]//Proc of the 49th Annual Int Symp on Computer Architecture. New York: ACM, 2022: 396–409

[34] Bhatla A, Navneet, Panda B. The Maya cache: A storage-efficient and secure fully-associative last-level cache[C]/Proc of the 51st Int Symp on Computer Architecture. New York: ACM, 2024: 32−44

[35] Wong L D, Wu Hao, Molder C, et al. Baleen: ML admission & prefetching for flash caches[C]//Proc of the 22nd USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2024: 347−371

[36] Liu Yubo, Ren Yuxin, Liu Mingrui, et al. Optimizing file systems on heterogeneous memory by integrating dram cache with virtual memory management[C]//Proc of the 22nd USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2024: 71−87

[37] McAllister S, Berg B, Tutuncu-Macias J, et al. Kangaroo: Caching billions of tiny objects on flash[C]//Proc of the 28th Symp on Operating Systems Principles. New York: ACM, 2023: 243−262

[38] Chen Jiayi, Sharma N, Khan T, et al. Darwin: Flexible learning-based CDN caching[C]//Proc of the 37th ACM Special Interest Group on Data Communication. New York: ACM, 2024: 981−999

[39] Yang Juncheng, Zhang Yazhuo, Qiu Ziyue, et al. FIFO queues are all you need for cache eviction [C]//Proc of the 29th Symp on Operating Systems Principles. New York: ACM, 2023: 130−149

[40] Yan Gang, Li Jian. Towards latency awareness for content delivery network caching[C]//Proc of the 2022 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2022: 789−804

[41] Mirheidari A S, Arshad S, Onarlioglu K, et al. Cached and confused: Web cache deception in the wild[C]//Proc of the 29th USENIX Security Symp. Berkeley, CA: USENIX Association, 2020: 665−682

[42] Ma Yun, Liu Xuanzhe, Mei Hong. Measurement and optimization of browser cache performance for mobile Web applications[J]. Journal of Software, 2020, 31(7): 1980−1996(in Chinese)
(马郓, 刘譞哲, 梅宏. 面向移动 Web 应用的浏览器缓存性能度量与优化[J]. 软件学报, 2020, 31(7): 1980−1996)

[43] Wang Huan, Wu Kui, Wang Jianping, et al. Rldish: Edge-assisted QoE optimization of HTTP live streaming with reinforcement learning[C]// Proc of the 43rd IEEE Conf on Computer Communications. Piscataway, NJ: IEEE 2020: 706−715

[44] Fuerst A, Sharma P. Faascache: Keeping serverless computing alive with greedy-dual caching[C]//Proc of the 26th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2021: 386−400

[45] Roy B R, Patel T, Tiwari D. Icebreaker: Warming serverless functions better with heterogeneity[C]//Proc of the 27th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2022: 753−767

[46] Li Guopeng, Tan Haisheng, Zhang Xuan, et al. Online container caching with late-warm for IoT data processing[C]//Proc of the 40th Int Conf on Data Engineering. Piscataway, NJ: IEEE 2024: 1547−1560

[47] Traverso S, Ahmed M, Garetto M, et al. Temporal locality in today's content caching: Why it matters and how to model it[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(5): 5−12

[48] Kwon W, Zhuohan Li, Zhuang Siyuan, et al. Efficient memory management for large language model serving with Paged Attention[C]//Proc of the 29th Symp on Operating Systems Principles. New York: ACM, 2023: 611−626

[49] Zhang Zhenyu, Sheng Ying, Zhou Tianyi, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models[C]//Advances in Neural Information Processing Systems. Cambridge, MA: MIT, 2023: 34661−34710

[50] Liu Yuhan, Li Hanchen, Cheng Yihua, et al. Cachegen: KV cache compression and streaming for fast language model serving[J]. arXiv preprint, arXiv: 2310.07240, 2023

[51] Agarwal S, Mitra S, Chakraborty S, et al. Approximate caching for efficiently serving text-to-image diffusion models[C]//Proc of the 21st USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2024: 1173−11189

[52] Ma Ziyu, Sun Bin, Li Shutao. A two-stage selective fusion framework for joint intent detection and slot filling[J]. IEEE Transactions on Neural Networks and Learning Systems, 2024, 35(3): 3874−3885

[53] NVIDIA. Intelligent virtual assistant [EB/OL]. [2024-05-21]. https://www.nvidia.com/en-us/ai-data-science/ai-workflows/intelligent-virtual-assistant/

[54] Kinza Y. Virtual assistant (ai assistant) [EB/OL]. [2024-05-21]. https://www.techtarget.com/searchcustomerexperience/definition/virtual-assistant-AI-assistant

[55] Google. bert-base-chinese [EB/OL]. [2024-04-01]. https://hugging-face.co/google-bert/bert-base-chinese

[56] MokaAI. M3E models [EB/OL]. [2024-04-01]. https://hugging-face.co/moka-ai/m3e-small

[57] Meta. FAISS. [EB/OL]. [2024-04-01]. https://ai.meta.com/tools/faiss/

[58] Patil G. Shishir, Zhang Tianjun, Fang Vivian, et al. Goex: perspectives and designs towards a runtime for autonomous llm applications[J]. arXiv preprint, arXiv: 2404.06921, 2024

[59] Chu A, Shoemaker C. Tutorial: Use code interpreter sessions in semantic kernel with azure container apps [EB/OL]. [2024-05-24]. https://learn.microsoft.com/en-us/azure/container-apps/sessions-tutorial-semantic-kernel

[60] LangChain. Security[EB/OL]. [2024-05-24]. https://python.lang-chain.com/v0.1/docs/security/

[61] Chinese information processing society of china. The evaluation of chinese human-computer dialogue technology [EB/OL]. [2024-02-01]. https://conference.cipsc.org.cn/smp2019/evaluation.html (in Chinese)
(中国中文信息学会. 中文人机对话技术评测 [EB/OL]. [2024-02-01]. https: //conference.cipsc.org.cn/smp2019/evaluation.html)

[62] Mehta S, Sekhavat M, Cao Q, et al. OpenELM: An efficient language model family with open training and inference framework [EB/OL]. [2024-05-01]. https://machinelearning.apple.com/research/openelm

[63] Beatty S. Tiny but mighty: The Phi−3 small language models with big potential [EB/OL]. [2024-05-01]. https://news.microsoft.com/source/features/ai/the-phi-3-small-language-models-with-big-potential/

[64] Lee S, Choi J, Lee J, et al. Explore, select, derive, and recall: Augmenting LLM with human-like memory for mobile task automation [J]. arXiv preprint, arXiv: 2312.03003, 2023

Li Guopeng, born in 1997. PhD candidate. His main research interests include edge intelligence, large language model-driven agent, and machine learning system.

李国鹏, 1997 年生, 博士研究生. 主要研究方向为边缘智能、大语言模型驱动的智能体、机器学习系统.

Wu Ruiqi, born in 2003. Master candidate. His main research interests include edge computing, large language model, and machine learning system.

吴瑞骐, 2003 年生. 硕士研究生. 主要研究方向为边缘计算、大语言模型、机器学习系统.

Tan Haisheng, born in 1981. PhD, professor. Member of CCF. His main research interests include edge intelligence, and system and networking for AI.

谈海生, 1981 年生. 博士, 教授. CCF 会员. 主要研究方向为边缘智能、人工智能系统与网络.

Chen Guoliang, born in 1938. Professor. Fellow of CCF. His main research interests include parallel algorithms, computer architectures, and computational intelligence.

陈国良, 1938 年生. 教授. CCF 会士. 主要研究方向为并行算法、计算机体系结构、计算智能.