中国科学 : 信息科学 2026 年 第 56 卷 第 6 期: 1532–1548
SCIENTIA SINICA Informationis
论文

《中国科学》杂志社
SCIENCE CHINA PRESS
CrossMark click for updates

面向依赖型任务的边缘智能协同计算卸载算法

姚枝秀[1], 李云[1,2], 夏士超[1*], 蒋溢[1], 梁吉申[2,3]

1. 重庆邮电大学计算机科学与技术学院, 重庆 400065
2. 重庆邮电大学通信与信息工程学院, 重庆 400065
3. 信息支援部队工程大学通信士官学校, 重庆 400035
* 通信作者. E-mail: xiashichao@cqupt.edu.cn

收稿日期: 2025–09–29; 修回日期: 2025–12–17; 接受日期: 2026–02–27; 网络出版日期: 2026–05–28

国家自然科学基金 (批准号: 62501093, 62301099)、中国博士后科学基金 (批准号: 2025MD774181)、重庆市优秀科学家人才项目 (批准号: cstc2024ycjh-bgzxm003)、重庆市自然科学基金 (批准号: CSTB2025NSCQ-LZX0077, CSTB2025NSCQ-LZX0052, CSTB2024NSCQ-QCXMX0063) 和重庆市教委科学技术研究计划项目青年项目 (批准号: KJQN202300638) 资助

摘要 人工智能与移动边缘计算的深度融合, 已成为实现 6G 网络智能内生愿景的重要驱动力, 其中智能计算卸载作为重要环节之一受到广泛关注. 针对多样化依赖型任务在边缘资源受限场景中的计算卸载问题, 提出了一种分布式智能协同计算卸载方法. 首先, 综合考虑边缘节点服务有限、任务多样性与执行依赖性等因素, 建立端边协同的计算卸载及多维资源分配优化问题模型, 目标是在服务容忍时延约束下最大化系统长期平均任务处理效用. 然后, 提出一种基于图注意力多智能体深度强化学习的依赖型任务计算卸载算法求解优化问题. 该算法在训练阶段采用图注意力网络挖掘全局状态的重要信息以学习高效协同策略参数, 在执行阶段由终端用户根据本地状态独立决策, 从而实现分布式智能协同计算卸载. 最后, 通过仿真实验验证所提算法性能. 仿真结果表明, 与现有的算法相比, 所提算法在不同网络参数下均具有最优性能.

关键词 边缘智能, 计算卸载, 依赖型任务, 协同计算, 多智能体深度强化学习

1 引言

随着物联网、人工智能 (artificial intelligence, AI) 和大数据等技术的发展, 涌现了大量以资源密集型和时延敏感型为主要特征的新兴应用, 例如人机交互、扩展现实和感官互联等. 为满足新兴应用的极致性能需求, 移动边缘计算 (mobile edge computing, MEC) 技术在网络边缘提供就近的云服务功能, 在高可靠低时延方面取得了一定进展 [1,2]. 其中, 计算卸载作为 MEC 的关键技术之一, 旨在通过设计合理的任务卸载决策与资源分配策略, 在缓解终端设备计算负担的同时保障任务性能需求, 已成为当前研究的热点方向.

近年来, 相关研究多聚焦于独立任务的计算卸载, 然而在实际应用中, 任务通常由多个相互依赖的子任务构成. 例如, 人脸识别任务通常包括对象获取、人脸检测、预处理、特征提取和分类等子任务 [3], 这些子任务之间存在严格的依赖关系, 如特征提取的输出需作为分类的输入. 针对依赖型任务的计算卸载问题, 部分研究 [4~7] 基于精确数学建模或先验全局知识, 采用传统的单阶段优化方法进行求解. 虽然此类方法在静态 MEC 场景下能够获得理论最优解, 但难以满足实际动态 MEC 网络中的序贯决策与实时自适应需求.

引用格式: 姚枝秀, 李云, 夏士超, 等. 面向依赖型任务的边缘智能协同计算卸载算法. 中国科学: 信息科学, 2026, 56: 1532–1548, doi: 10.1360/SSI-2025-0416
Yao Z X, Li Y, Xia S C, et al. Intelligent collaborative computation offloading for dependent tasks in edge networks. Sci Sin Inform, 2026, 56: 1532–1548, doi: 10.1360/SSI-2025-0416

© 2026《中国科学》杂志社 www.scichina.com infocn.scichina.com
https://www.sciengine.com/doi/10.1360/SSI-2025-0416

深度强化学习 (deep reinforcement learning, DRL) 等 AI 方法具备强大的推理与学习能力, 能够有效应对序贯决策问题并适应动态环境的变化. 与此同时, 6G 的发展愿景强调网络智能内生化, 推动着 MEC 在边缘智能方向演进. 在此背景下, 基于 AI 算法模型的 MEC 优化方法被视为实现网络智能内生的关键一环. 目前, 一些工作 [8~16] 采用 DRL 等 AI 方法研究依赖型任务的计算卸载问题. 然而, 上述工作普遍假设边缘服务器具备完备的服务能力, 忽视了实际场景中部分任务所需服务未部署的情况. 实际上, 由于业务请求的随机性、节点功能定位和资源受限等因素, 边缘服务器通常采取按需部署服务的策略. 在此情况下, 任务的多样化服务需求进一步加剧了服务匹配的难度. 例如, 在人脸识别应用中, 特征提取必须在部署了相关服务的处理器上才能执行 [17]. 因此, 单一边缘节点难以保障任务的可靠性和低时延要求.

多节点协同处理任务能够扩展单一节点的功能, 实现资源互补与服务能力提升. 近年来, 已有研究在服务约束下提出依赖型任务的协同计算方法 [17~23]. 然而, 这些方法局限于单阶段优化 [17~21, 23] 或只考虑单用户场景 [18, 19, 22], 难以应对多用户、多边缘节点环境中的复杂动态特性. 因此, 在任务依赖和资源受限场景中, 如何实现协同计算卸载及资源分配的分布式智能自适应推理、优化与决策, 仍是急需解决的核心挑战.

针对上述挑战, 本文提出了一种基于图注意力多智能体深度强化学习的依赖型任务计算卸载算法 (graph attention-based multi-agent deep reinforcement learning for dependent task offloading, GMA-Dep). 该算法面向多用户、多边缘节点 MEC 系统, 充分考虑任务执行依赖性与服务资源受限条件, 以实现分布式智能协同计算卸载及资源分配策略. 本文的主要贡献如下.

(1) 在多用户多边缘节点的动态 MEC 场景下, 综合考虑单节点资源受限、任务多样性与执行依赖性, 构建了面向依赖型任务的多节点协同计算卸载框架. 该框架支持用户间的端端协同、边缘节点间的边边协同以及端边间的跨层协同计算, 以全面挖掘边缘侧与空闲端侧的可用资源. 在此基础上, 以最大化所有用户的长期平均任务处理效用为目标, 建立了计算卸载与多维资源分配优化问题模型.

(2) 本文提出一种 GMA-Dep 算法求解上述优化问题. 该算法采用集中式训练与分布式执行框架, 在集中式训练阶段, 终端用户采用图注意力网络提取对协同决策更有价值的邻居用户特征, 从而更高效地探索和学习全局最优的协同计算卸载及资源分配策略模型参数; 在分布式执行阶段, 每个终端用户基于其本地状态信息, 实现计算卸载及资源分配的分布式智能决策.

(3) 为验证 GMA-Dep 算法的性能, 开展了大量数值仿真实验. 仿真结果表明, 与现有算法相比, 所提 GMA-Dep 算法在不同 MEC 场景参数下均表现出最优性能, 验证了该算法能够针对依赖型任务, 有效协调终端用户和边缘节点进行多节点智能协同计算.

2　系统模型

如图 1 所示, 本文考虑若干用户设备 (user equipment, UE) 与一个部署了 MEC 服务器的基站相关联, UE 集合定义为 $\mathcal{M}=\{1,2,\cdots,M\}$, 其中, M 表示 UE 的数量. 对于每个 UE, 其所关联的基站定义为本地基站 (local base station, LBS). 此外, 系统中存在若干配置了 MEC 服务器的邻居基站 (neighbor base station, NBS) 用于协同处理计算任务, 可实现资源互补和边缘计算能力增强. NBS 集合定义为 $\mathcal{N}_{\mathrm{NBS}}=\{\mathrm{NBS}_1,\mathrm{NBS}_2,\cdots,\mathrm{NBS}_N\}$, 其中, N 表示 NBS 的数量. MEC 系统的时间离散为长度相等的时隙集合 $\mathcal{T}=\{0,1,2,\cdots\}$, 每个 UE 在每个时隙 $t\in\mathcal{T}$ 根据参数为 p 的伯努利 (Bernoulli) 分布生成一个任务, 如人脸识别、手势识别等. 定义 $\xi_m(t)\in\{0,1\}$ 表示 UE_m ($m\in\mathcal{M}$) 在时隙 t 的任务请求状态, 其中, $\xi_m(t)=1$ 表示 UE_m 在时隙 t 请求了一个任务, $\xi_m(t)=0$ 表示 UE_m 在时隙 t 没有生成任务, 处于空闲状态. UE 之间支持设备到设备 (device-to-device, D2D) 通信. 如图 1 所示, UE 请求的任务可以在本地设备、附近空闲 UE、LBS 或 NBS 计算, 也可由上述节点协同计算.

2.1　依赖型任务模型

本文考虑 UE 请求的任务由数个具有依赖关系的子任务组成, 如人脸识别. 普遍的人脸识别通常由 5 个子任务组成, 分别是图像采集、人脸检测、预处理、特征提取和分类. 当执行人脸识别应用时, 需要考虑子任务之间的

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

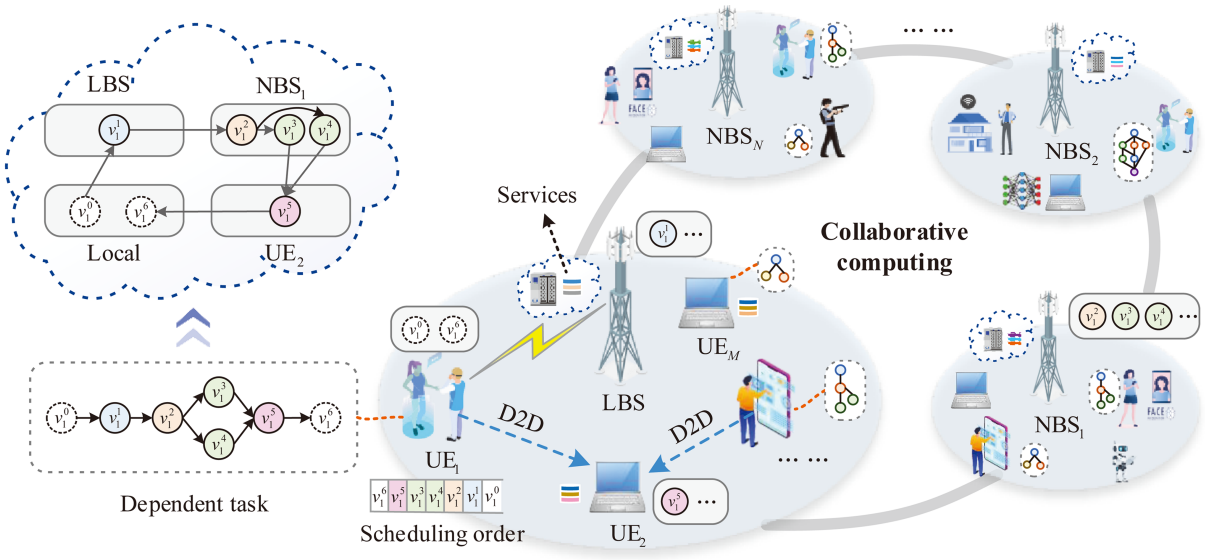


图 1　(网络版彩图) 系统模型.

Figure 1　(Color online) System model.

依赖性, 例如 “特征提取” 的输出是 “分类” 的输入, 即 “分类” 需要接收到 “特征提取” 的输出结果后才能计算. 此外, 考虑到子任务类型的多样性, 任务的计算需要处理器配置和该任务类型相关的数据 (程序代码和数据库). 如 “特征提取” 只能在配置了相关机器学习模型的处理器上计算. 定义系统中任务类型集合为 $\mathcal{K}=\{1,2,\cdots,K\}$, 其中, $K=|\mathcal{K}|$ 表示子任务类型的数量.

不失一般性, 依赖型任务可用有向无环图 (directed acyclic graph, DAG) 描述 [8,9]. 具体地, UE_m 在时隙 t 请求的任务表示为 $G_m(t)=(\mathcal{V}_m(t),\mathcal{E}_m(t))$, 其中, $\mathcal{V}_m(t)=\left\{v_m^0(t),v_m^1(t),\cdots,v_m^{|\mathcal{V}_m|+1}(t)\right\}$ 表示子任务集合, $v_m^0(t)$ 和 $v_m^{|\mathcal{V}_m|+1}(t)$ 表示虚拟的开始任务和退出任务; $e\left(v_m^i(t),v_m^j(t)\right)\in\mathcal{E}_m(t)$ 表示边集, 用于描述子任务之间的执行依赖关系, 其中, $e\left(v_m^i(t),v_m^j(t)\right)=1$ 表示第 i 个子任务 $v_m^i(t)$ 是第 j 个子任务 $v_m^j(t)$ 的前驱任务, 即 $v_m^j(t)$ 需要接收到 $v_m^i(t)$ 的计算结果作为输入数据才能开始执行, 否则, $e\left(v_m^i(t),v_m^j(t)\right)=0$.

对于子任务 $v_m^i(t)$, 本文采用 4 个参数进行描述, 定义为 $v_m^i(t)=\left\{d_m^i(t),\rho_m^i(t),\psi_m^i(t),\tau_m^{\max}(t)\right\}$, 其中, $d_m^i(t)$ 表示数据量大小, $\rho_m^i(t)$ 表示子任务所需的 CPU 周期数, $\psi_m^i(t)\in\mathcal{K}$ 表示子任务类型, $\tau_m^{\max}(t)$ 表示任务 $G_m(t)$ 的任务处理容忍时延. 对于虚拟子任务 $v_m^0(t)$ 和 $v_m^{|\mathcal{V}_m|+1}(t)$, 数据大小设置为 0. 子任务间需要满足严格的执行顺序, 即某些子任务必须在其前驱任务完成后方可执行. 定义 UE_m 的任务调度顺序为 $q_m(t)=\left(q_m^1(t),q_m^2(t),\cdots,q_m^{|\mathcal{V}_m(t)|}(t)\right)$, 其中, $q_m^i(t)\in\{1,2,\cdots,|\mathcal{V}_m(t)|\}$ 表示子任务 $v_m^i(t)$ 的调度优先级, $q_m^i(t)$ 值越小, 表明调度优先级越高. 调度顺序 $q_m(t)$ 满足子任务之间的执行依赖关系, 在此条件下, 一个任务可能具有多个可行调度顺序, 不同的调度顺序具有不同的服务时延. 由于子任务间的依赖性, 为便于分析, 定义开始时间 ST 表示子任务能够开始执行的最早时间, 完成时间 FT 表示子任务计算完成的时间.

2.2　通信模型

在蜂窝基站覆盖的上行通信场景中, UE 通过无线链路向 LBS 传输任务, 同时支持 UE 间 D2D 通信以实现近距离端端协同计算. UE 的上行通信与 D2D 通信均采用正交频分多址接入方式, 子信道带宽分别定义为 B^{up} 和 B^{D2D}. 根据香浓 (Shannon) 公式, 可得 UE_m 发送子任务 $v_m^i(t)$ 到 LBS 的上行传输速率, 以及发送子任务 $v_m^i(t)$ 到 $\mathrm{UE}_{m'}$ 的数据传输速率分别为

$$R_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)=B^{\mathrm{up}}\log_2\left(1+\frac{p_m^i(t)\,g_{m,\mathrm{LBS}}(t)}{\delta^2}\right),\tag{1}$$

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

$$R_{m,m'}^{i,\mathrm{D2D}}(t)=B^{\mathrm{D2D}}\log_2\left(1+\frac{p_m^i(t)\,g_{m,m'}(t)}{\delta^2}\right),\tag{2}$$

其中, $p_m(t)$ 表示 UE_m 在时隙 t 的发射功率; $g_{m,\mathrm{LBS}}(t)$ 和 $g_{m,m'}(t)$ 分别表示 UE_m 在时隙 t 与 LBS 和 $\mathrm{UE}_{m'}$ 的信道增益; δ^2 表示背景噪声功率.

2.3　计算模型

任意子任务 $v_m^i(t)$ 具有以下计算模式: (1) 本地设备计算; (2) 通过上行无线链路卸载到 LBS 计算; (3) 通过 D2D 通信链路卸载到附近 UE 进行端端协同计算; (4) 由 LBS 转发至附近 NBS 进行边边协同计算. 对于 LBS 计算和边边协同计算模式, 鉴于边缘网络中服务部署需综合考虑地域业务特征、节点功能定位, MEC 服务器通常根据任务特征与服务分布策略, 仅缓存系统中部分任务类型所需的服务. 定义 $\Phi_n(t)\subseteq\mathcal{K}$ 表示基站 n ($n\in\{\mathrm{LBS}\}\cup\mathcal{N}_{\mathrm{NBS}}$) 在时隙 t 缓存的服务集. 若 UE_m 在时隙 t 生成计算请求, 则子任务 $v_m^i(t)$ 在时隙 t 可卸载的边缘节点集如下:

$$\mathcal{N}_m^{i,\mathrm{avl}}(t)=\left\{n\,\middle|\,\xi_m(t)=1,\psi_m^i(t)\in\Phi_n(t)\right\}_{\forall n\in\mathrm{LBS}\cup\mathcal{N}_{\mathrm{NBS}}}.\tag{3}$$

处于空闲状态且具有相应服务的 UE 可以为附近 UE 提供计算服务. 定义 UE_m 的服务集为 $\Lambda_m\subseteq\mathcal{K}$, 则 $v_m^i(t)$ 可卸载的 UE 集如下:

$$\mathcal{M}_m^{i,\mathrm{avl}}(t)=m\cup\left\{x\,\middle|\,\xi_m(t)=1,\xi_x(t)=0,\psi_m^i(t)\in\Lambda_x\right\}_{\forall x\in\mathcal{M}}.\tag{4}$$

综上, 若 UE_m 在时隙 t 生成计算请求, 可定义子任务 $v_m^i(t)$ 的卸载决策变量为 $\alpha_m^i(t)\in\mathcal{M}_m^{i,\mathrm{avl}}(t)\cup\mathcal{N}_m^{i,\mathrm{avl}}(t)$, 其中, $\alpha_m^i(t)=m$ 表示子任务在本地设备计算; $\alpha_m^i(t)=\mathrm{LBS}$ 表示子任务卸载到 LBS 计算; $\alpha_m^i(t)=\mathrm{NBS}_n$ 表示子任务通过 LBS 转发到 NBS_n 计算; $\alpha_m^i(t)=m'$ ($m'\neq m$) 表示子任务通过 D2D 链路卸载到附近 $\mathrm{UE}_{m'}$ 计算. 综上可得 UE_m 请求的计算任务在时隙 t 的卸载决策变量为 $\alpha_m(t)=\left(\alpha_m^0(t),\alpha_m^1(t),\cdots,\alpha_m^{|\mathcal{V}_m|+1}(t)\right)$. 需要说明的是, 本文中虚拟的开始任务 $v_m^0(t)$ 和退出任务 $v_m^{|\mathcal{V}_m|+1}(t)$ 在本地设备计算, 因此, $\alpha_m^0(t)=m$, $\alpha_m^{|\mathcal{V}_m|+1}(t)=m$.

2.3.1　本地设备计算模型

如果子任务 $v_m^i(t)$ 在本地设备计算, 即 $\alpha_m^i(t)=m$, 则 $v_m^i(t)$ 首先在本地设备等待执行. 当任务达到可执行条件时, UE_m 将提供 CPU 计算资源计算该任务. 由于子任务之间的执行依赖性, 某些子任务需要接收到其所有前驱任务的计算结果后才能开始执行. 考虑到计算结果的数据量较小, 本文忽略计算结果的传输时间和能耗 [24]. 因此, 当 $v_m^i(t)$ 在本地设备执行时, 其开始时间 $ST_{m,m}^i(t)$ 是其前驱任务计算完成时间, 表示如下:

$$ST_{m,m}^i(t)=\max_{v_m^j(t)\in pred[v_m^i(t)]}\left\{FT_{m,\alpha_m^j}^j(t)\right\},\tag{5}$$

其中, $pred\left[v_m^i(t)\right]$ 表示 $v_m^i(t)$ 的前驱任务集; $FT_{m,\alpha_m^j}^j(t)$ 表示前驱任务 $v_m^j(t)$ 在其目标节点执行时的完成时间.

当子任务满足执行条件时, UE_m 将提供 CPU 计算资源执行该任务. 定义 $f_{m,m}^i(t)$ 表示 UE_m 在时隙 t 分配给子任务 $v_m^i(t)$ 的计算资源, 则 $v_m^i(t)$ 在本地设备执行的计算时延为

$$T_{m,m}^{i,\mathrm{comp}}(t)=\frac{d_m^i(t)\,\rho_m^i(t)}{f_{m,m}^i(t)}.\tag{6}$$

子任务 $v_m^i(t)$ 在本地设备执行的计算能耗为

$$E_{m,m}^{i,\mathrm{comp}}(t)=\kappa_m\left(f_{m,m}^i(t)\right)^3T_{m,m}^{i,\mathrm{comp}}(t),\tag{7}$$

其中, κ_m 表示与 UE_m 芯片架构相关的有效能量系数.

子任务 $v_m^i(t)$ 在本地设备计算时的结束时间 $FT_{m,m}^i(t)$ 是其计算完成的最早时间, 表示如下:

$$FT_{m,m}^i(t)=ST_{m,m}^i(t)+T_{m,m}^{i,\mathrm{comp}}(t).\tag{8}$$

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

2.3.2　本地基站计算模型

如果子任务 $v_m^i(t)$ 在本地基站计算, 即 $\alpha_m^i(t)=\mathrm{LBS}$, UE_m 首先将 $v_m^i(t)$ 通过无线链路传输至 LBS. 当子任务达到可执行条件时, 由 LBS 侧配置的 MEC 服务器提供计算资源执行该子任务. UE_m 卸载子任务 $v_m^i(t)$ 到 LBS 的上行传输时延和能耗分别为

$$T_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)=\frac{d_m^i(t)}{R_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)}, \tag{9}$$

$$E_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)=p_m(t)\,T_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t). \tag{10}$$

$v_m^i(t)$ 在 LBS 执行时的开始时间 $ST_{m,\mathrm{LBS}}^i(t)$ 是其所有前驱任务完成时间和数据传输完成时间的最大值, 计算如下:

$$ST_{m,\mathrm{LBS}}^i(t)=\max\left\{\max_{v_m^j(t)\in pred[v_m^i(t)]}\left\{FT_{m,\alpha_m^j}^j(t)\right\},T_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)+AT_m^i(t)\right\}, \tag{11}$$

其中, $AT_m^i(t)$ 表示 UE_m 发送 $v_m^i(t)$ 时的通信资源可用时间, 计算如下:

$$AT_m^i(t)=\sum_{v_m^j(t)\in q_m[v_m^i(t)]}\left[\frac{d_m^j(t)}{R_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)}\cdot\mathbf{1}_{\left\{\alpha_m^j(t)\in\mathcal{N}_m^{j,\mathrm{avl}}(t)\right\}}+\frac{d_m^j(t)}{R_{m,m'}^{i,\mathrm{D2D}}(t)}\cdot\mathbf{1}_{\left\{\alpha_m^j(t)=m'\right\}}\right], \tag{12}$$

其中, $q_m\left[v_m^i(t)\right]$ 表示基于子任务调度顺序 $q_m(t)$ 在 $v_m^i(t)$ 之前调度的任务集合; $\mathbf{1}_{\{\cdot\}}$ 为指示函数, 如果事件 $\{\cdot\}$ 为真, 则 $\mathbf{1}_{\{\cdot\}}=1$, 否则 $\mathbf{1}_{\{\cdot\}}=0$.

$v_m^i(t)$ 上传至 LBS 且满足执行条件后, MEC 服务器将提供计算资源执行该任务. 定义 f_{LBS} 表示 LBS 提供的计算资源, 可得 $v_m^i(t)$ 在 LBS 执行时的计算时延为

$$T_{m,\mathrm{LBS}}^{i,\mathrm{comp}}(t)=\frac{d_m^i(t)\,\rho_m^i(t)}{f_{\mathrm{LBS}}}. \tag{13}$$

子任务 $v_m^j(t)$ 的结束时间 $FT_{m,\mathrm{LBS}}^i(t)$ 是其计算完成的最早时间, 表示如下:

$$FT_{m,\mathrm{LBS}}^i(t)=ST_{m,\mathrm{LBS}}^i(t)+T_{m,\mathrm{LBS}}^{i,\mathrm{comp}}(t). \tag{14}$$

2.3.3　协同计算模型

(1) 端端协同计算模型. 如果子任务 $v_m^i(t)$ 在 $\mathrm{UE}_{m'}$ 计算, 即 $\alpha_m^i(t)=m'$ $(m'\in\mathcal{M}_m^{i,\mathrm{avl}}(t)\setminus\{m\})$. 此时, UE_m 基于 D2D 通信将 $v_m^i(t)$ 传输至空闲设备 $\mathrm{UE}_{m'}$. $v_m^i(t)$ 发送至 $\mathrm{UE}_{m'}$ 的数据传输时延和能耗分别为

$$T_{m,m'}^{i,\mathrm{D2D}}(t)=\frac{d_m^i(t)}{R_{m,m'}^{i,\mathrm{D2D}}(t)}, \tag{15}$$

$$E_{m,m'}^{i,\mathrm{D2D}}(t)=p_m(t)\,T_{m,m'}^{i,\mathrm{D2D}}(t). \tag{16}$$

子任务 $v_m^i(t)$ 在 $\mathrm{UE}_{m'}$ 执行时的开始时间 $ST_{m,m'}^i(t)$ 为

$$ST_{m,m'}^i(t)=\max\left\{\max_{v_m^j(t)\in pred[v_m^i(t)]}\left\{FT_{m,\alpha_m^j}^j(t)\right\},T_{m,m'}^{i,\mathrm{D2D}}(t)+AT_m^i(t)\right\}. \tag{17}$$

当获得所有前驱任务的计算结果后, $\mathrm{UE}_{m'}$ 将提供计算资源 $f_{m,m'}^i(t)$ 执行 $v_m^i(t)$, 则 $v_m^i(t)$ 在空闲设备 $\mathrm{UE}_{m'}$ 执行的计算时延和能耗分别为

$$T_{m,m'}^{i,\mathrm{comp}}(t)=\frac{d_m^i(t)\,\rho_m^i(t)}{f_{m,m'}^i(t)}, \tag{18}$$

$$E_{m,m'}^{i,\mathrm{comp}}(t)=\kappa_{m'}\left(f_{m,m'}^i(t)\right)^3T_{m,m'}^{i,\mathrm{comp}}(t). \tag{19}$$

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

子任务 $v_m^i(t)$ 在 $\mathrm{UE}_{m'}$ 执行时的结束时间 $FT_{m,m'}^i(t)$ 为

$$FT_{m,m'}^i(t)=ST_{m,m'}^i(t)+T_{m,m'}^{i,\mathrm{comp}}(t). \tag{20}$$

(2) 边边协同计算模型. 如果 $v_m^i(t)$ 在 NBS 计算, 即 $\alpha_m^i(t)=\mathrm{NBS}_n$ ($\mathrm{NBS}_n\in\mathcal{N}_m^{i,\mathrm{avl}}(t)\setminus\{\mathrm{LBS}\}$), UE_m 首先将 $v_m^i(t)$ 发送至 LBS, 然后由 LBS 转发到 NBS_n, 则 UE_m 卸载 $v_m^i(t)$ 到 NBS_n 时的上传能耗为 $E_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)$. 定义 LBS 到 NBS_n 的平均传输速率为 $R_{\mathrm{LBS},\mathrm{NBS}_n}$, 则 $v_m^i(t)$ 的上传时延为

$$T_{m,\mathrm{NBS}_n}^{i,\mathrm{up}}(t)=T_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)+\frac{d_m^i(t)}{R_{\mathrm{LBS},\mathrm{NBS}_n}}. \tag{21}$$

子任务 $v_m^i(t)$ 在 NBS_n 执行时的开始时间 $ST_{m,\mathrm{NBS}_n}^i(t)$ 为

$$ST_{m,\mathrm{NBS}_n}^i(t)=\max\left\{\max_{v_m^j(t)\in pred[v_m^i(t)]}\left\{FT_{m,\alpha_m^j}^j(t)\right\},T_{m,\mathrm{NBS}_n}^{i,\mathrm{up}}(t)+AT_m^i(t)\right\}. \tag{22}$$

当子任务传输至 NBS_n 且满足执行条件后, 由 NBS_n 配置的 MEC 服务器分配计算资源 f_{NBS_n} 执行该任务, 则 $v_m^i(t)$ 在 NBS_n 的计算时延为

$$T_{m,\mathrm{NBS}_n}^{i,\mathrm{comp}}(t)=\frac{d_m^i(t)\rho_m^i(t)}{f_{\mathrm{NBS}_n}}. \tag{23}$$

进一步, 子任务 $v_m^i(t)$ 在 NBS_n 执行时的结束时间 $FT_{m,\mathrm{NBS}_n}^i(t)$ 为

$$FT_{m,\mathrm{NBS}_n}^i(t)=ST_{m,\mathrm{NBS}_n}^i(t)+T_{m,\mathrm{NBS}_n}^{i,\mathrm{comp}}(t). \tag{24}$$

3　问题建模

根据第 2 节的系统模型, 可得 UE_m 的服务时延为虚拟退出任务的结束时间, 计算如下:

$$T_m(t)=FT_{m,m}^{|\mathcal{V}_m|+1}(t)\cdot\mathbf{1}_{\{\xi_m(t)=1\}}, \tag{25}$$

其中, $FT_{m,m}^{|\mathcal{V}_m|+1}(t)$ 表示虚拟退出任务的结束时间 (虚拟任务在本地设备计算); $\mathbf{1}_{\{\xi_m(t)=1\}}$ 判断 UE_m 在时隙 t 的任务请求状态, 本文定义处于空闲状态的 UE_m 的服务时延为 0.

对于具有计算请求的 UE, 其能耗主要包括子任务本地计算能耗, 以及卸载子任务时的数据传输能耗; 对于空闲 UE, 其能耗主要来自处理其他 UE 子任务的计算能耗, 则 UE_m 在时隙 t 能耗表示为

$$\begin{aligned}E_m(t)=&\sum_{v_m^i(t)\in\mathcal{V}_m(t)}\Big[E_{m,m}^{i,\mathrm{comp}}(t)\cdot\mathbf{1}_{\{\alpha_m^i(t)=m\}}+E_{m,\mathrm{LBS}}^{i,\mathrm{up}}(t)\cdot\mathbf{1}_{\left\{\alpha_m^i(t)\in N_m^{i,\mathrm{avl}}(t)\right\}}\\&+E_{m,m'}^{i,\mathrm{D2D}}(t)\cdot\mathbf{1}_{\left\{\alpha_m^i(t)\in\mathcal{M}_m^{i,\mathrm{avl}}(t)\setminus\{m\}\right\}}\Big]\cdot\mathbf{1}_{\{\xi_m(t)=1\}}\\&+\sum_{m'\in\mathcal{M}\setminus\{m\}}\sum_{v_{m'}^i(t)\in\mathcal{V}_{m'}(t)}\left[E_{m',m}^{i,\mathrm{comp}}(t)\cdot\mathbf{1}_{\left\{\alpha_{m'}^i(t)=m\right\}}\right]\cdot\mathbf{1}_{\{\xi_m(t)=0,\xi_{m'}(t)=1\}}.\end{aligned} \tag{26}$$

为刻画 MEC 系统的资源利用效率, 定义时隙 t 的任务处理能效为

$$EE(t)=\frac{\sum\limits_{m\in\mathcal{M}}\mathbf{1}_{\{\tau_m^{\max}(t)-T_m(t)\geqslant 0,\xi_m(t)=1\}}}{\sum\limits_{m\in\mathcal{M}}E_m(t)}. \tag{27}$$

$EE(t)$ 用于衡量单位能耗完成的任务数量, 如果所有 UE 在时隙 t 均处于空闲状态, 此时, 设置 $EE(t)=0$. 在此基础上, 为权衡任务服务时延和能效, 本文定义 MEC 系统在时隙 t 处理任务的效用函数如下:

$$U(t)=\lambda\cdot EE(t)-(1-\lambda)\cdot\sum_{m\in\mathcal{M}}T_m(t), \tag{28}$$

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

其中, λ 表示时延和能效的权重系数.

本文旨在子任务执行依赖约束、可用服务节点、UE 可用传输功率和计算资源的约束下, 通过优化子任务调度顺序、子任务卸载策略, 以及 UE 的发射功率分配策略和计算资源分配策略, 最大化系统任务处理的长期平均效用. 优化问题建模如下:

$$\max_{\boldsymbol{q},\boldsymbol{\alpha},\boldsymbol{p},\boldsymbol{f}} \quad \lim_{|\mathcal{T}|\to\infty} \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} U(t) \tag{29}$$

$$\text{s.t.} \quad q_m^i(t) \in \{0,1,\cdots,|\mathcal{V}_m(t)|+1\}, \forall m,i,t, \tag{C1}$$

$$\alpha_m^i(t) \in \mathcal{M}_m^{i,\text{avl}}(t) \cup \mathcal{N}_m^{i,\text{avl}}(t), \forall m,i,t, \tag{C2}$$

$$T_m(t) \leqslant \tau_m^{\max}(t), \forall m,i,t, \tag{C3}$$

$$p_m^{\min} \leqslant p_m^i(t) \leqslant p_m^{\max}, \forall m,i,t, \tag{C4}$$

$$\sum_{m'\in\mathcal{M}} \sum_{v_{m'}^i(t)\in\mathcal{V}_{m'}(t)} f_{m',m}^i(t)\cdot \mathbf{1}_{\left\{\alpha_{m'}^i(t)=m\right\}} \leqslant F_m(t), \forall m,i,t, \tag{C5}$$

其中, $\boldsymbol{q}=\{q_m(t)\}_{\forall m,t}$ 表示子任务调度决策; $\boldsymbol{\alpha}=\left\{\alpha_m^i(t)\right\}_{\forall m,i,t}$ 表示子任务卸载决策; $\boldsymbol{p}=\left\{p_m^i(t)\right\}_{\forall m,i,t}$ 表示 UE_m 的传输功率分配决策; $\boldsymbol{f}=\left\{f_{m,m}^i(t), f_{m,m'}^i(t)\right\}_{\forall m,m',i,t}$ 表示 UE_m 的计算资源分配决策; 约束条件 (C2) 表示子任务只能卸载到配置了相应服务功能的节点执行; 约束条件 (C3) 表示任务的服务时延不能超过其容忍时延; 约束条件 (C4) 表示 UE 的传输功率分配范围在最小值 $p_m^{\min}$ 和最大值 $p_m^{\max}$ 之间; 约束条件 (C5) 表示 UE 的计算资源分配不能超过其最大可用计算资源 $F_m(t)$. 优化问题 (29) 是一个序贯决策问题, 通过寻找每个时隙最优的 $\boldsymbol{q}$, $\boldsymbol{\alpha}$, $\boldsymbol{p}$ 和 $\boldsymbol{f}$ 可以最大化 MEC 系统中所有 UE 的长期平均任务处理效用.

4　算法设计

在优化问题 (29) 中, 单阶段优化方法难以刻画系统的时序动态特性. 同时, 该问题属于混合整数非线性规划, 采用传统优化方法往往面临求解复杂度高、难以获得可行解等困难. 集中式优化方法在获取全局状态信息方面存在困难, 还可能因单点失效而导致可靠性不足. 因此, 本文提出了一种基于图注意力多智能体深度强化学习的依赖型任务计算卸载算法 (graph attention-based multi-agent deep reinforcement learning for dependent task offloading, GMA-Dep) 求解优化问题. 首先, 将优化问题 (29) 转化为分布式部分可观测马尔科夫决策过程 (decentralized partially observable Markov decision process, Dec-POMDP); 然后, 给出了 GMA-Dep 的算法框架和详细原理.

4.1　Dec-POMDP 问题转化

在优化问题 (29) 中, 未来的 UE 可用计算资源状态、UE 空闲状态、UE 任务信息状态、信道状态、MEC 服务器的服务集合状态, 以及子任务调度、计算卸载及资源分配状态均与历史时隙状态无关, 即计算卸载过程具有马尔科夫性质. 因此, 本文将原始优化问题转化为 Dec-POMDP 问题, 其中, 每个 UE 作为一个智能体. Dec-POMDP 可描述为一个元组 $\langle\{\mathcal{O}_m\}_{\forall m},\{\mathcal{A}_m\}_{\forall m},R,\gamma\rangle$, 其中, $\mathcal{O}_m$ 表示智能体 UE_m 的本地观测空间; $\mathcal{A}_m$ 表示智能体 UE_m 的动作空间; R 表示奖励函数; $\gamma\in[0,1)$ 表示折扣因子. 智能体 UE_m 在时隙 t 接收到一个本地观测值 $o_m(t)\in\mathcal{O}_m$, 选取卸载动作 $a_m(t)\in\mathcal{A}_m$, 同时将 $a_m(t)$ 输入到一个子任务调度模块, 子任务调度模块基于所选 $a_m(t)$ 返回智能体 UE_m 的调度顺序 $q_m(t)$. 然后, 智能体 UE_m 基于 $q_m(t)$ 执行卸载动作 $a_m(t)$, 同时根据奖励函数 R 返回奖励 $r_m(t)$, 并将本地观测转移到下一个状态 $o_m(t+1)$. 每个智能体的最终目的是探索改进其策略 μ_m, 使得长期累积回报 $\sum_{t\in\mathcal{T}}\gamma^i r_n(t+i+1)$ 最大化. 接下来定义本文 Dec-POMDP 的本地观测、动作和奖励函数.

(1) 本地观测. 对于具有计算请求的智能体 UE_m ($\xi_m(t)=1$), 每个时隙的本地观测包括本地任务信息状态、本地信道状态、本地和空闲 UE 的计算资源状态、系统设备空闲状态、系统服务集合状态. 对于空闲 UE, 则无

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

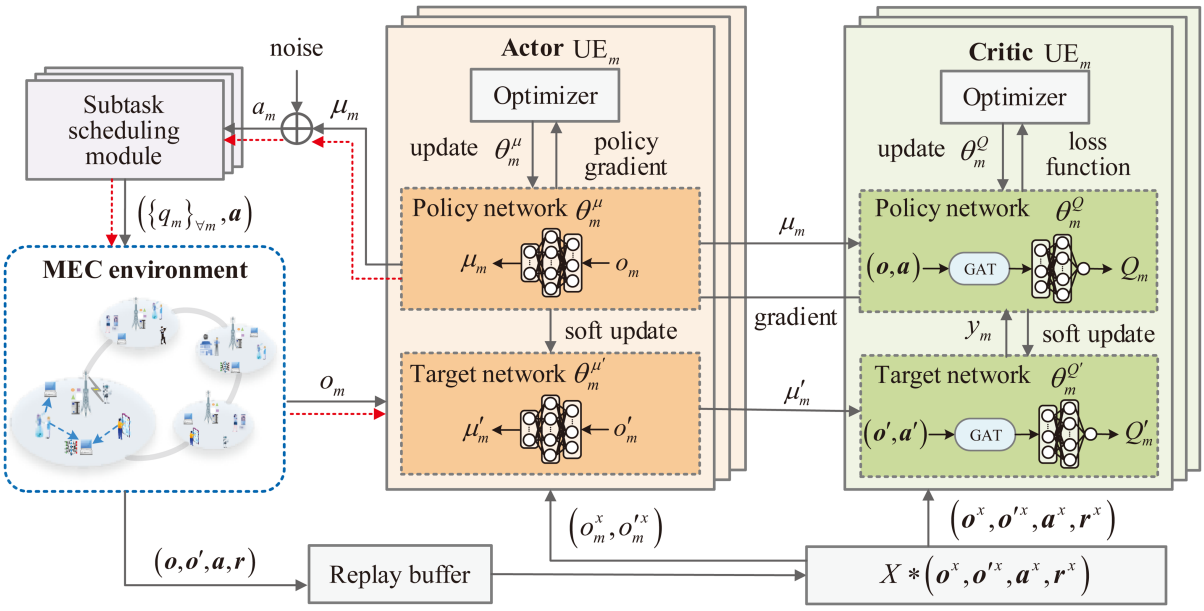


图 2　(网络版彩图) GMA-Dep 算法框架.

Figure 2　(Color online) The framework of GMA-Dep algorithm.

需观测状态信息. 据此定义智能体 UE$_m$ 在时隙 t 的本地观测 $o_m(t) \in \mathcal{O}_m$ 如下:

$$o_m(t) = \Big\{ G_m(t),\ g_{m,\mathrm{LBS}}(t),\ g_{m,m'}(t), \\ \{F_m(t),\ F_{m'}(t)\}_{\forall m',\, \xi_{m'}(t)=0}, \{\xi_m(t)\}_{\forall m}, \{\Phi_n(t)\}_{\forall n} \Big\}_{\xi_m(t)=1}, \tag{30}$$

其中, $G_m(t)$ 为 UE$_m$ 的任务请求信息, 包括任务完成容忍时延 $\tau_m^{\max}(t)$、子任务之间的执行依赖关系、子任务的数据量大小 $d_m^i(t)$、每 bit 数据所需的 CPU 周期数 $\rho_m^i(t)$ 和子任务类型 $\psi_m^i(t)$.

(2) 动作. 根据优化问题 (29) 和本文定义的 Dec-POMDP 过程, 对于具有计算请求的智能体 UE$_m$, 在时隙 t 的动作包括子任务卸载、传输功率分配, 以及计算资源分配动作, 其中, 计算资源分配包括本地计算资源分配和向空闲 UE 请求的协同计算资源分配. 对于空闲 UE 则不采取动作. 据此定义 UE$_m$ 的动作 $a_m(t) \in \mathcal{A}_m$ 如下:

$$a_m(t) = \left\{ \alpha_m^i(t), p_m^i(t), f_{m,m}^i(t), f_{m,m'}^i(t) \right\}_{\xi_m(t)=1, \forall i, m' \in \mathcal{M}_m^{i,\mathrm{avl}}(t) \setminus \{m\}}. \tag{31}$$

(3) 奖励函数. 根据优化问题 $\mathcal{P}2$, 多智能体协同优化的目标是在 UE 服务时延约束下最大化 MEC 系统任务处理效用. 因此, 本文定义智能体 UE$_m$ 在时隙 t 的奖励函数为

$$r_m(t) = U(t) + \sum_{m \in \mathcal{M}} H\left(\tau_m^{\max}(t) - T_m(t)\right) \cdot \mathbf{1}_{\{\xi_m(t)=1\}}, \tag{32}$$

其中, $H\left(\tau_m^{\max}(t) - T_m(t)\right)$ 表示是否满足 UE 服务容忍时延的分段函数, $\tau_m^{\max}(t) - T_m(t) \geqslant 0$ 时 $H(\cdot) = 1$, 表示服务时延在容忍时延约束内; 否则 $H(\cdot) = -1$.

4.2　基于 GMA-Dep 的计算卸载及资源分配算法

GMA-Dep 算法框架如图 2 所示, 其中, 每个 UE 智能体都配置了一个图注意力增强的 DDPG 网络和一个子任务调度模块. GMA-Dep 算法采用 CTDE 框架, 在集中式训练阶段, 给定智能体 UE$_m$ 当前时隙下的本地观测 o_m, Actor 网络将根据当前策略 $\mu_m(o_m)$ 采取动作 a_m. 接着, 智能体 UE$_m$ 将动作 a_m 输入其子任务调度模块, 子任务调度模块输出智能体的子任务调度顺序 q_m. 智能体 UE$_m$ 将调度顺序 q_m 和计算卸载及资源分配动作 a_m 与 MEC 环境进行交互, 并根据奖励函数获得奖励 r_m, 同时将本地观测转移到下一个状态 o'_m. 所有智能

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

体将样本存入经验回放池中形成一个共享样本, 经验回放池包含序列 $(\boldsymbol{o}, \boldsymbol{o}', \boldsymbol{a}, \boldsymbol{r})$, 其中, $\boldsymbol{o}$ 表示所有智能体的观测信息; $\boldsymbol{o}'$ 表示所有智能体在下一个时隙的本地观测; $\boldsymbol{a}$ 表示所有智能体的动作; $\boldsymbol{r}$ 表示所有智能体的奖励. 然后, 基于图注意力网络 (graph attention network, GAT) 的 Critic 网络从经验回放池中进行采样, 通过一个集中式动作价值函数来评估 Actor 网络采取的动作, 从而引导 Actor 网络学习最优策略. 接下来, 阐述子任务调度模块、基于图注意力增强的 Critic 网络, 以及 GMA-Dep 算法训练过程.

4.2.1　子任务调度模块

智能体 UE_m 根据当前策略执行子任务卸载决策、传输功率分配决策和计算资源分配决策等动作, 并将动作输入子任务调度模块. 子任务调度模块根据子任务的处理时间成本, 以及内部执行依赖关系获得调度优先级. 对于子任务 v_m^i, 其处理时间成本为

$$\begin{aligned} T_m^i(t) = {} & T_{m,m}^{i,\mathrm{comp}} \cdot \mathbf{1}_{\{\alpha_m^i = m\}} + \left(T_{m,m'}^{i,\mathrm{D2D}} + T_{m,m'}^{i,\mathrm{comp}}(t)\right) \cdot \mathbf{1}_{\left\{\alpha_m^i(t) \in \mathcal{M}_m^{i,\mathrm{avl}} \setminus \{m\}\right\}} \\ & + \left(T_{m,\alpha_m^i}^{i,\mathrm{up}} + T_{m,\alpha_m^i}^{i,\mathrm{comp}}\right) \cdot \mathbf{1}_{\left\{\alpha_m^i \in \mathcal{N}_m^{i,\mathrm{avl}}\right\}}. \end{aligned} \tag{33}$$

根据任务处理时间成本以及内部执行依赖关系, 定义子任务 v_m^i 的调度优先级为

$$priority\left(v_m^i\right) = T_m^i + \max_{v_m^j \in succ[v_m^i]} \left\{priority\left(v_m^j\right)\right\}, \tag{34}$$

其中, $succ\left[v_m^i\right]$ 表示 v_m^i 的后续任务集. 如果 v_m^i 没有后续任务, 则 $priority\left(v_m^i\right) = T_m^i$. 然后, 对各个子任务的调度优先级进行排序, $priority\left(v_m^i\right)$ 越大, 表明 v_m^i 调度优先级越高, q_m^i 的值越小, 即 $q_m^i = |\{j \in \mathcal{V}_m | priority(v_m^j) > priority(v_m^i)\}|$. 需要说明的是, 虚拟开始任务的优先级最高, 即 $q_m^0 = 0$, 虚拟退出任务的优先级最低, 即 $q_m^{|\mathcal{V}_m|+1} = |\mathcal{V}_m| + 1$. 对于空闲 UE 则不用计算子任务调度顺序. 分布式子任务调度模块与多智能体强化学习通过交替迭代的方式实现联合优化, 每个 UE 智能体将调度顺序、计算卸载、传输功率分配、计算资源分配决策与 MEC 环境进行交互并返回奖励.

4.2.2　图注意力增强的 Critic 网络

在基于 D2D 通信的边缘计算系统中, 具有计算请求的 UE 可将子任务卸载至附近空闲 UE 执行. 然而, 由于 UE 的随机任务请求与移动行为, UE 的空闲状态与网络拓扑均呈现时空变化特征. 因此, 在制定计算卸载与资源分配策略时, 应根据当前时隙的状态信息, 自适应地建模 UE 间的关联关系, 以获得更有效的协同计算决策. 为此, 本文在 CTDE 框架下采用基于 GAT 的 Critic 网络, 通过注意力机制提取对决策更有价值的邻居节点特征, 弱化无关节点特征带来的噪声影响. 例如, 系统中的 UE 智能体需要协同制定联合动作 (计算卸载和资源分配策略) 以最大化系统收益, 因此, 需更加关注非空闲 UE 的行为. GAT 的特征提取能力能够强化 Critic 对全局状态的表示能力, 从而提升动作价值估计的准确性, 以引导 Actor 网络生成更加稳定、高效的卸载与资源分配策略.

假设 MEC 系统状态在单个时隙内处于准静态, 因此, 本文将多 UE 智能体在单个时隙内的特征构造为一个无向图 $\mathcal{G}(\mathcal{M}, \mathcal{E})$, 其中, $\mathcal{M}$ 为 UE 智能体的集合; $\mathcal{E}$ 表示 UE 之间边的集合. 如果 UE_m 和 $\mathrm{UE}_{m'}$ 有边则表示 UE_m 和 $\mathrm{UE}_{m'}$ 在当前时隙都请求了任务; 如果 UE 处于空闲状态, 则与其他 UE 之间不构成边. 由于 UE 任务请求状态的随机性, $\mathcal{E}$ 会随时隙动态变化. 据此定义一个由边集 $\mathcal{E}$ 确定的邻居节点集合 $\mathcal{B}_m \subseteq \mathcal{M}$, 有边的 UE 互为邻居节点. 需要注意的是, 由于 UE 计算请求状态和移动行为的随机性, UE_m 的邻居节点集合将呈现时变特征.

基于 GAT 的 Critic 网络将每个 UE_m 的本地观测和动作作为图 $\mathcal{G}$ 中第 m 个节点的特征输入 GAT 网络, 并输出各节点的隐藏表示 $(h_1, h_2, \cdots, h_M)$. 具体地, GAT 首先计算 UE_m 与其邻居 UE_k $(k \in \mathcal{B}_m)$ 之间的注意力系数, 具体计算为 $e_{m,k} = att\left(\mathbf{W}(o_m, a_m), \mathbf{W}(o_k, a_k)\right)$, 其中, $att(\cdot)$ 表示注意力机制, 是由 LeakyReLU 函数激活的单层前馈神经网络; $\mathbf{W}$ 表示学习参数. 注意力系数 $e_{m,k}$ 表示 UE_m 的特征对其邻居 UE_k 的重要性. 本文只

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

考虑 UE_m 的一阶邻居 (包括节点 UE_m). 然后, 采用 softmax 函数对注意系数进行归一化, 计算如下:

$$\beta_{m,k}=\mathrm{softmax}_k\left(e_{m,k}\right)=\frac{\exp\left(e_{m,k}\right)}{\sum\limits_{n\in\mathcal{B}_m}\exp\left(e_{m,n}\right)}. \tag{35}$$

为提高训练过程的稳定性, 采用多头注意力机制. 在获得归一化注意力系数后, Critic UE_m 的输出隐藏表示由 L 个独立的注意力头计算, 表示为

$$h'_m=\Big\|_{l=1}^{L}\sigma\left(\sum_{k\in\mathcal{B}_m}\beta_{m,k}^{l}\mathbf{W}^l h_k\right), \tag{36}$$

其中, $\|$ 表示拼接操作; l 表示注意力头的序号. 最后, 如图 2 所示, Critic 网络将 $(h_1,h_2,\cdots,h_M)$ 输入 MLP 网络, 并输出集中式 $Q_m\left(\boldsymbol{o},\boldsymbol{a}\right)$.

4.2.3　GMA-Dep 算法训练过程

定义 $\mu=\{\mu_m\}_{\forall m}$ 和 $Q=\{Q_m\}_{\forall m}$ 分别表示所有智能体的 Actor 网络和 Critic 网络集合, 对应参数分别为 $\theta^{\mu}=\{\theta_m^{\mu}\}_{\forall m}$ 和 $\theta^{Q}=\left\{\theta_m^{Q}\right\}_{\forall m}$. 对于 Actor 网络, 任意智能体 UE_m 通过策略梯度的方式更新策略, 策略梯度如下所示:

$$\nabla_{\theta_m^{\mu}}\mathcal{J}\approx\frac{1}{X}\sum_{x=1}^{X}\nabla_{\theta_m^{\mu}}\mu_m\left(o_m^x;\theta_m^{\mu}\right)\nabla_{a_m}Q_m\left(\boldsymbol{o}^x,\boldsymbol{a}^x;\theta_m^{Q}\right)\big|_{a_m=\mu_m\left(o_m^x;\theta_m^{\mu}\right)}, \tag{37}$$

其中, X 表示从经验回放池中随机抽样的小批量样本数量; x 表示第 x 个样本; $Q_m\left(\boldsymbol{o}^x,\boldsymbol{a}^x;\theta_m^{Q}\right)$ 表示由参数 θ_m^{Q} 描述的集中式动作价值函数.

对于 Critic 网络, 智能体 UE_m 通过最小化损失函数的方式更新对应参数 θ_m^{Q}, 损失函数表示如下:

$$Loss=\frac{1}{X}\sum_{x=1}^{X}\Big[\left(y_m^x-Q_m\left(\boldsymbol{o}^x,\boldsymbol{a}^x;\theta_m^{Q}\right)\right)^2\Big], \tag{38}$$

其中, $y_m^x=r_m^x+\gamma Q'_m\left(\boldsymbol{o'}^x,\boldsymbol{a'}^x;\theta_n^{Q'}\right)\Big|_{a'_m=\mu'_m\left(o'^x_m;\theta_n^{\mu'}\right)}$; μ'_m 表示由参数 $\theta_m^{\mu'}$ 描述的目标 Actor 网络; Q'_m 表示由参数 $\theta_m^{Q'}$ 描述的目标 Critic 网络. 最后, 每个智能体以软更新的方式更新目标 Actor 网络和目标 Critic 网络的参数, 分别表示如下:

$$\theta_m^{\mu'}=\varsigma\theta_m^{\mu}+\left(1-\varsigma\right)\theta_m^{\mu'}, \tag{39}$$

$$\theta_m^{Q'}=\varsigma\theta_m^{Q}+\left(1-\varsigma\right)\theta_m^{Q'}, \tag{40}$$

其中, ς 表示更新率.

在集中式训练过程中, 基于 GAT 的 Critic 网络考虑了 MEC 系统的全局状态及所有 UE 的任务卸载与资源分配动作, 每个 UE 在统一的全局动作价值函数 Q_m 的评估下实现策略协同更新. 具体地, UE 智能体在训练阶段可共享彼此的本地观测与动作信息, 并利用 GAT 网络挖掘 UE 间的协作关系, 强化 Critic 网络的特征表示能力. 该协同学习机制使得每个 UE 在策略更新时能够充分考虑其他 UE 的卸载和资源分配决策, 同时, 充分感知网络状态变化、空闲 UE 和边缘服务器等节点的可用资源等情况, 从而形成稳定性更强、协同性更高的协同计算卸载和资源分配策略.

当模型参数训练结束后, 如图 2 所示, 在分布式执行阶段 (红色虚线过程), 每个 UE 智能体只需要获得本地观测信息, 并将其输入相应 Actor 网络中即可获得学习到的计算卸载和资源分配动作. GMA-Dep 的训练过程如算法 1 所示.

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

算法 1 The training process of GMA-Dep.

Initialize: the θ^{μ} of policy actor network, the $\theta^{\mu'}$ of target actor network, the θ^{Q} of policy critic network, the $\theta^{Q'}$ of target critic network, and replay buffer $\mathcal{D}$;

for $episode = 1, 2, \ldots, T_e$ **do**
  Receive initial local observation $\boldsymbol{o}$;
  for $t = 1, 2, \ldots, T_s$ **do**
    Each agent UE_m selects action $a = \mu_m(o_m)$;
    Calculate q_m by subtask scheduling module;
    Execute $\boldsymbol{a}$ and q_m, while observing reward $\boldsymbol{r}$ and next observation o'_m;
    Store $(\boldsymbol{o}, \boldsymbol{o}', \boldsymbol{a}, \boldsymbol{r})$ in replay buffer $\mathcal{D}$;
    for agent $\mathrm{UE}_m = \mathrm{UE}_1, \mathrm{UE}_2, \ldots, \mathrm{UE}_M$ **do**
      Sample a random mini-batch of X samples $(\boldsymbol{o}^x, \boldsymbol{o}'^x, \boldsymbol{a}^x, \boldsymbol{r}^x)$ form $\mathcal{D}$;
      Update policy actor network by (37);
      Update policy critic network by (38);
      $\theta_m^{\mu'} = \varsigma\theta_m^{\mu} + (1-\varsigma)\,\theta_m^{\mu'}$;
      $\theta_m^{Q'} = \varsigma\theta_m^{Q} + (1-\varsigma)\,\theta_m^{Q'}$;
    end for
  end for
end for

4.2.4　算法复杂度分析

在集中式训练阶段, GMA-Dep 的时间复杂度主要取决于 UE 智能体的数量 M、训练回合数 T_e、每个回合时间步数 T_s、小批量样本数量 X, 以及 Actor 和 Critic 网络结构和子任务调度模块. 对于 Actor 网络, 采用两层全连接 MLP, 其单步时间复杂度为 $O\left(Md_{\mathrm{in}}^{\mathrm{a}}h^{\mathrm{a}}\right)$, 其中, $d_{\mathrm{in}}^{\mathrm{a}}$ 和 h^{a} 分别表示输入的本地观测维度和隐藏层维度; 对于 Critic 网络, 采用一层 GAT 网络对全局状态和动作进行特征提取, 并用两层全连接 MLP 输出 Q 值, 其单步时间复杂度为 $O\left(L\left(Md_{\mathrm{in}}^{\mathrm{c}}h^{c} + |\mathcal{E}|\,h^{c}\right) + Mh^{c}h'^{c}\right)$, 其中, $|\mathcal{E}|$ 表示 UE 之间边的集合; $d_{\mathrm{in}}^{\mathrm{c}}$, h^{c} 和 h'^{c} 分别表示输入的全局状态 – 动作维度、GAT 输出的隐藏维度和 MLP 的隐藏维度. 对于子任务调度模块, 其单步时间复杂度计算为 $O\left(M\left|\mathcal{E}_m\right|\right)$, 其中, $|\mathcal{E}_m|$ 为 DAG 任务的边数量. 综上分析, GMA-Dep 在训练时的时间复杂度为

$$O\left[T_eT_sX\left(Md_{\mathrm{in}}^{\mathrm{a}}h^{\mathrm{a}} + L\left(Md_{\mathrm{in}}^{\mathrm{c}}h^{c} + |\mathcal{E}|\,h^{c}\right) + Mh^{c}h'^{c}\right) + T_eT_sM\left|\mathcal{E}_m\right|\right]. \tag{41}$$

当模型训练结束后, 每个 UE 智能体基于 Actor 网络和子任务调度模块执行动作, 其时间复杂度为 $O[d_{\mathrm{in}}^{\mathrm{a}}h^{\mathrm{a}} + |\mathcal{E}_m|]$.

5　仿真与分析

本小节进行了仿真实验以验证所提 GMA-Dep 算法的性能, 对比算法如下.

(1) no-coop [25]. no-coop 不考虑 UE 间及 BS 间的协同计算, 子任务只在本地设备或 LBS 计算.

(2) no-E2E [16]. no-E2E 不考虑边边协同计算, 子任务只在本地设备、空闲设备或 LBS 计算.

(3) no-End2End [8]. no-End2End 不考虑端端协同计算, 子任务只在本地设备、LBS 或 NBS 计算.

(4) 基于单智能体深度强化学习的依赖型任务计算卸载算法 (singe-agent deep reinforcement learning-based computation offloading for dependent tasks, SA-Dep) [11]. SA-Dep 算法中每个智能体都配置了一个 DDPG 网络和一个子任务调度模块, 并根据其本地观测独立进行训练和执行.

(5) 基于多智能体深度强化学习的依赖型任务计算卸载算法 (multi-agent deep reinforcement learning-based computation offloading for dependent tasks, MA-Dep). MA-Dep 算法基于 CTDE 框架, 与 GMA-Dep 不同的是, 在策略参数训练阶段, MA-Dep 采用基于 MLP 的 Critic 网络输出 Q 值.

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

需要说明的是, no-coop, no-E2E, no-End2End 和 SA-Dep 算法的 Actor 网络和 Critic 网络均基于 MLP 训练. no-coop, no-E2E, no-End2End 仅在计算卸载模式上与 MA-Dep 算法不同, 其他方面均完全一致.

5.1　仿真环境和参数设置

本仿真采用 Pytorch 1.13 深度学习框架和 Adam 优化器实现 GMA-Dep 和所有对比算法. 所有实验都基于 Windows 10 操作系统 (CPU: Intel(R) Core(TM) i9-10920X CPU @ 3.5 GHz, GPU: NVIDIA GeForce RTX 3090) 完成. 本文 MEC 仿真场景包括一个 LBS 以及 2 个 NBS, 共有 5 个 UE 在其 LBS 覆盖范围内随机移动, 基站的覆盖半径为 200 m. 设置上行无线信道带宽为 20 MHz; D2D 信道带宽为 20 MHz; 信道增益设置为 $(d(t))^{-3}$, 其中, $d(t)$ 为发射机和接收机之间的距离; UE 发射功率范围设置为 5∼33 dBm; 基站发射功率设置为 1 W; 背景噪声功率设置为 −50 dBm. 基站之间的传输速率设置为 50 Mbps; 每个 UE 的可用计算资源设置为 4.5∼5.5 GHz, 有效能量系数设置为 10^{-27} W×s^3/cycle^3; MEC 服务器在每个时隙提供给任意到达任务的计算资源设置为 3 GHz; 设置 MEC 系统中任务类型数量为 20 个, 每个 MEC 服务器随机包含 8 种服务类型.

每个 UE 在每个时隙以参数为 0.5 的伯努利分布生成一个 DAG 任务, 子任务类型数量设置为 6. 本文采用 DAG 任务合成器生成多样化任务 [26]. DAG 合成器的参数包括 n, fat 和 $density$, 其中, n 控制 DAG 任务的子任务数量; fat 控制 DAG 的宽度和高度; $density$ 控制边的数量; 仿真中子任务数量默认设置为 $n = 5$; fat 和 $density$ 均从 {0.4, 0.5, 0.6, 0.7, 0.8} 随机选取; 每个子任务的数据量大小均匀分布在 [0.1, 0.5] Mbit 之间; 每种任务类型每 bit 所需 CPU 周期数均匀分布在 [1000, 1200] cycles/bit 之间; 每个任务的容忍时延在 $n\times$ [0.4, 0.7] s 之间随机选择; 时延和能效的权重系数设置为 0.5;

对于所有基于深度强化学习的任务卸载算法, 设置折扣因子 $\gamma = 0.99$; Actor 网络的学习率为 0.0001; Critic 网络的学习率设置为 0.001; 经验回放池大小为 5×10^5; 小批量样本数量为 $X = 256$; 更新率为 $\varsigma = 0.005$; 训练回合数为 4000, 每个回合时间步为 100; 探索概率在 1×10^4 个时间步内从 1 下降到 0.05. 在 GMA-Dep 算法中, Actor 网络采用两层全连接 MLP 结构. Critic 网络采用一层具有 8 个注意力头的 GAT, 其输出隐藏维度设置为 32. 注意力机制采用由 LeakyReLU 激活函数激活的单层前馈神经网络, 其中, 负斜率系数设为 0.2. 然后通过两层全连接 MLP 输出对应的动作价值. 对于所有对比算法, Actor 网络与 Critic 网络均统一采用包含两层全连接 MLP 结构. 需要注意的是, 除非另有说明, 上述参数设置是每个实验的默认值.

5.2　仿真结果和性能分析

仿真实验主要从系统效用、能效、时延和任务完成率等方面进行验证, 其中, 时隙 t 的任务完成率定义为容忍时延内成功计算的 UE 数量与请求任务的 UE 数量之比, 如果时隙 t 没有任务请求, 即所有 UE 处于空闲状态, 则设置任务完成率为 1. 实验首先对比了不同算法的收敛性; 然后, 给出了用户设备数量、邻居基站数量、时延 – 能效权重系数、子任务数量, 以及任务请求概率等参数对不同算法性能的影响.

5.2.1　算法收敛性

图 3 给出了不同算法的系统奖励、系统效用和任务完成率跟随训练回合数增加的收敛过程. 从图中可以看出, 随着训练回合数的增加, 所有算法均能收敛到一个稳定值. 与其他基准算法相比, 本文所提 GMA-Dep 具有最优的性能. no-coop 的子任务只能在本地设备或 LBS 计算. 然而, 由于 LBS 只提供部分服务集合, 部分子任务只能在时延和能耗成本较高且计算资源稀缺的本地设备计算, 因此, 系统效用和任务完成率最低. no-E2E 不考虑边边协同计算, 但利用了空闲 UE 的资源; no-End2End 没有利用空闲 UE 的资源, 但可以将子任务卸载到邻居边缘节点计算, 两种算法均有效提升了系统效用和任务完成率. SA-Dep 虽然同时采用了端端协同和边边协同计算机制, 但该算法中每个 UE 智能体进行分布式独立训练, 导致智能体无法有效识别奖励变化是由自身的动作还是其他智能体的动作引起的, 从而难以学习到全局最优的计算卸载和资源分配策略. 相比之下, MA-Dep 和 GMA-Dep 采用了 CTDE 学习框架, 智能体在训练过程中考虑了 MEC 系统全局状态以及其他智能体的卸载动作, 从而形成稳定性更强、协同性更高的协同计算卸载和资源分配策略. 此外, GMA-Dep 采用 GAT 网络提取对

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

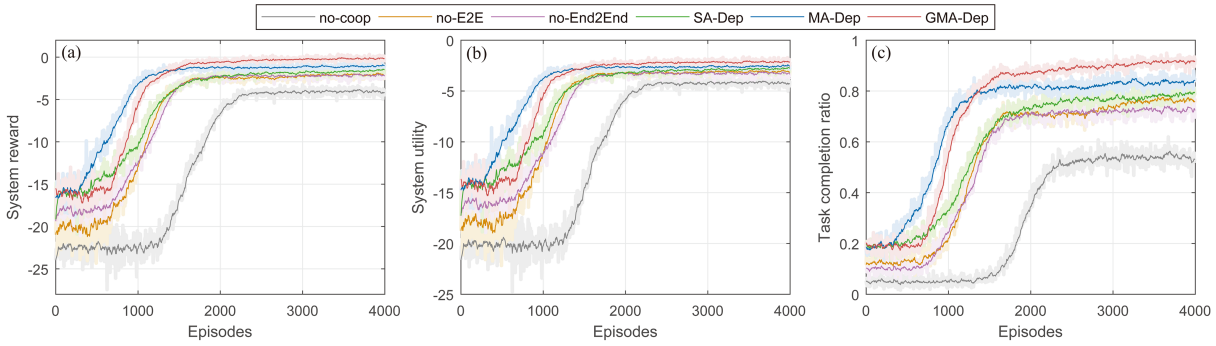


图 3　不同算法收敛过程. (a) 系统奖励; (b) 系统效用; (c) 任务完成率.

Figure 3　Convergence process of different algorithms. (a) System reward; (b) system utility; (c) task completion ratio.

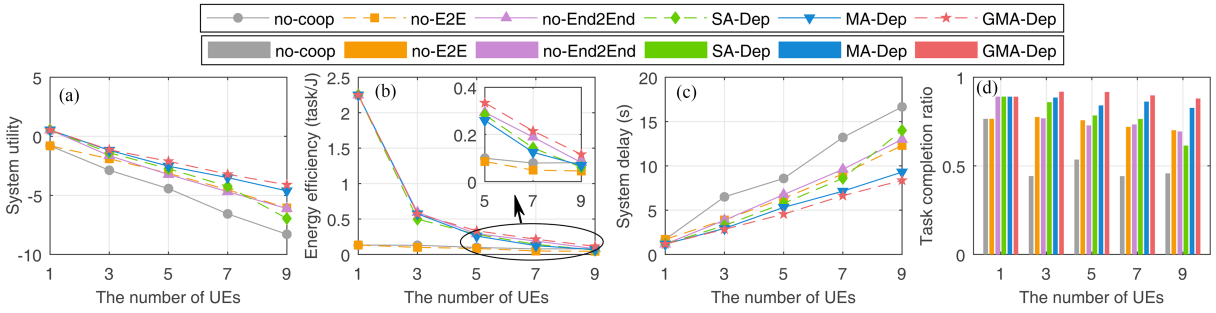


图 4　用户设备数量对不同算法的影响. (a) 系统效用; (b) 能效; (c) 系统时延; (d) 任务完成率.

Figure 4　Impact of the number of UEs on different algorithms. (a) System utility; (b) energy efficiency; (c) system delay; (d) task completion ratio.

协同策略更有价值的全局信息, 有效提升了动作价值估计的准确性. 与 no-coop, no-E2E, no-End2End, SA-Dep 和 MA-Dep 相比, GMA-Dep 的系统效用平均分别提升了 52.28%, 32.53%, 34.66%, 23.11%, 16.45%, 任务完成率平均分别提升了 71.27%, 21.12%, 25.93%, 16.93%, 9.05%. 验证了所提算法能够更有效地协调空闲设备和边缘节点进行多节点协同计算, 实现资源互补和能力增强.

5.2.2　用户设备数量对不同算法性能影响

为了验证所提算法在多用户 MEC 场景下的协同决策能力, 图 4 给出了不同 UE 数量下各算法的性能表现. 实验中 UE 数量设置为 1~9 个, 其他参数均为 5.1 小节中的默认值. 从图 4 中可以看出, 随着 UE 数量的增加, 所有算法的系统时延随之增加, 能效随之降低, 根据式 (28) 中效用的定义, 系统效用也随之降低. 与基准算法相比, 本文所提 GMA-Dep 在多 UE 场景下均具有最优的系统效用、能效、时延和任务完成率. 具体地, 与 no-coop, no-E2E, no-End2End 相比, 基于端边多节点协同计算的 MA-Dep 和 GMA-Dep 的性能具有显著性能优势. SA-Dep 虽然也采用了多节点协同计算, 但其完全分布式独立训练的方式无法应对 UE 智能体数量较大时训练环境的不稳定性问题, 因此, 性能在 UE 数量较大时急剧降低. 与 SA-Dep 相比, 基于 CTDE 学习框架的 MA-Dep 和 GMA-Dep 能够有效应对多用户 MEC 场景下的计算卸载和资源分配问题, 而 GMA-Dep 基于 GAT 更有效地学习了系统状态和动作信息, 因此, 具有最优算法性能.

5.2.3　邻居基站数量对不同算法性能影响

为评估所提算法在不同边缘节点规模下的性能优势, 图 5 给出了不同邻居基站数量下各算法的性能表现. 实验中 NBS 数量设置为 0~4 个, 其他参数均为 5.1 小节中的默认值. 观察图 5 可以发现, 由于 no-E2E 和 no-coop 没有考虑边缘节点之间的协同计算, 因此, 算法性能不受 NBS 数量变化的影响. 而 no-End2End, SA-Dep,

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

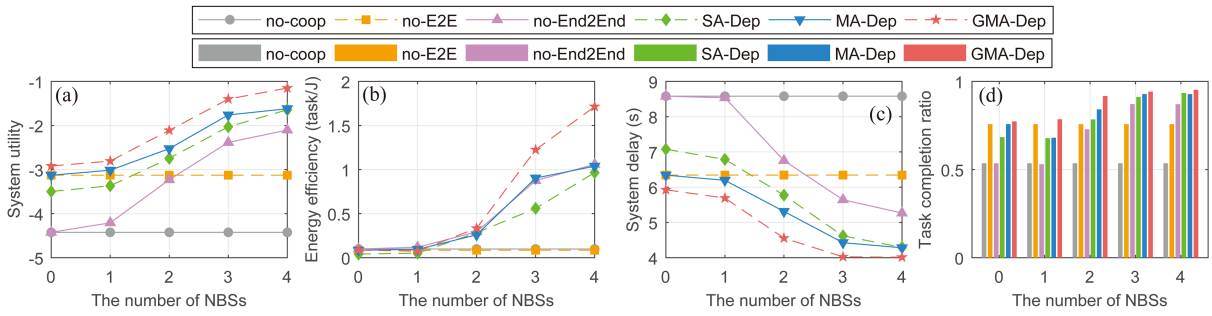


图 5　邻居基站数量对不同算法的影响. (a) 系统效用; (b) 能效; (c) 系统时延; (d) 任务完成率.

Figure 5　Impact of the number of NBSs on different algorithms. (a) System utility; (b) energy efficiency; (c) system delay; (d) task completion ratio.

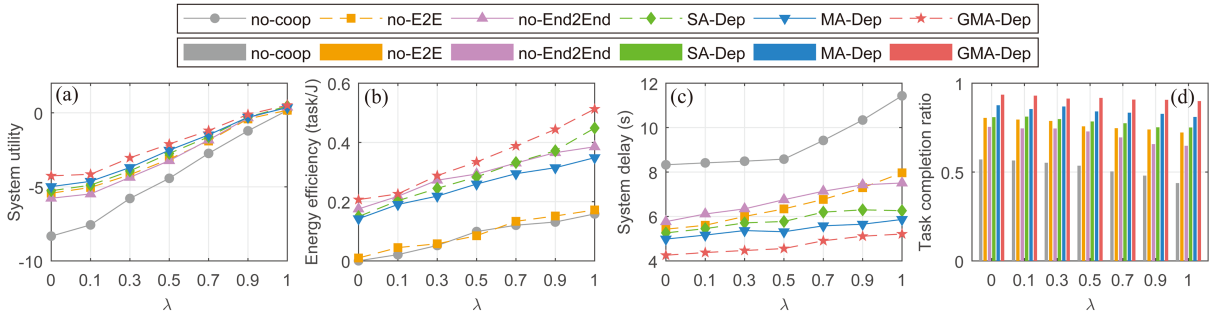


图 6　时延 – 能耗权重系数对不同算法的影响. (a) 系统效用; (b) 能效; (c) 系统时延; (d) 任务完成率.

Figure 6　Impact of λ on different algorithms. (a) System utility; (b) energy efficiency; (c) system delay; (d) task completion ratio.

MA-Dep 和 GMA-Dep 随着 NBS 数量的增加, 系统时延随之降低, 系统效用、能效和任务完成率随之增加. 这是由于 NBS 数量的增加扩展了系统的多样化任务服务能力, UE 请求的任务在计算时延较低的边缘节点执行的概率增加, 从而降低服务时延和本地设备能耗. 此外, 如图 5 所示, 不考虑边边协同计算的算法 (no-E2E 和 no-coop) 与考虑边边协同计算的算法 (no-End2End, SA-Dep, MA-Dep 和 GMA-Dep) 之间的性能差距随着 NBS 数量的增加而增加. 具体表现为可协同的 NBS 数量越多, 基于边边协同计算的算法性能优势越显著, 这进一步验证了边边协同计算的必要性和有效性. 与基准算法相比, 本文所提 GMA-Dep 在不同 NBS 数量下均具有最优的性能.

5.2.4　时延 – 能效权重系数对不同算法性能影响

时延 – 能效权重系数反映了 MEC 系统在时延性能与能效性能之间的权衡, 图 6 给出了时延 – 能效权重系数对各算法性能的影响. 实验中时延 – 能效权重系数设置为 0~1, 其他参数均为 5.1 小节中的默认值. 由图 6 可见, 随着权重系数的增大, 效用函数中能效项的权重逐渐提高、时延项的权重相应降低, 使得算法在决策过程中更加侧重于提升能效. 因此, 系统能效随权重系数的增加而逐步提升, 而由此带来的代价是系统时延的增加和任务完成率的降低. 综合上述影响, 在能效增益占主导作用的情况下, 系统效用整体随权重系数的增加而提高. 与基准算法相比, 本文所提 GMA-Dep 算法在不同时延 – 能效权重系数下均具有最优系统效用、能效、时延和任务完成率.

5.2.5　任务请求概率对不同算法性能影响

在实际的 MEC 场景中, UE 的计算请求具有随机性, 为验证随机请求场景下本文所提算法的性能, 图 7 给出了 UE 不同任务请求概率对各算法性能影响. 实验中每个 UE 在每个时隙以参数 0.1~1 的伯努利分布生成一个 DAG 任务, 其他参数均为 5.1 小节中的默认值. 从图 7 中可以观察到, 随着任务请求概率的增加, 所有算法的系统效用, 能效和任务完成率均随之降低, 而系统时延随之增加. 这是由于系统中的可用资源保持不变, 而计算

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

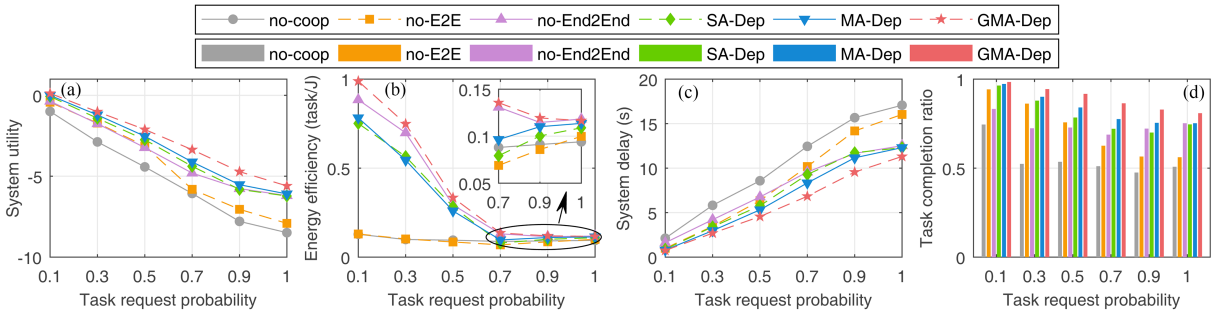


图 7　任务请求概率对不同算法的影响. (a) 系统效用; (b) 能效; (c) 系统时延; (d) 任务完成率.

Figure 7　Impact of request probability on different algorithms. (a) System utility; (b) energy efficiency; (c) system delay; (d) task completion ratio.

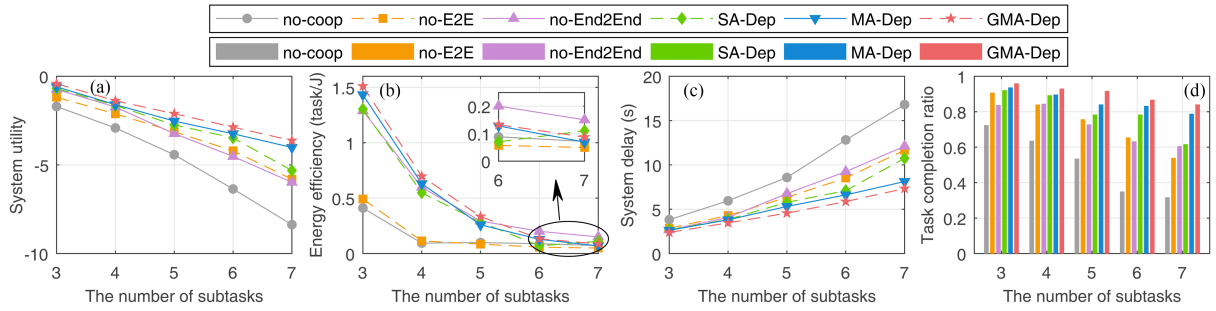


图 8　子任务数量对不同算法的影响. (a) 系统效用; (b) 能效; (c) 系统时延; (d) 任务完成率.

Figure 8　Impact of the number of subtasks on different algorithms. (a) System utility; (b) energy efficiency; (c) system delay; (d) task completion ratio.

任务请求不断增加, 从而导致有限资源的竞争加剧, 系统整体性能随之下降. 此外, 从图 7 可以观察到, 在任务请求概率较小 (0.1, 0.2) 和较大 (0.9, 1) 时, MA-Dep 与 SA-Dep 的性能差距相对较小. 当任务请求概率较小时, 系统中大多数 UE 处于空闲状态, 任务总量有限, 卸载决策复杂度较低, 两种算法均能够较好地协调空闲 UE 参与协同计算. 随着任务请求概率的增大, 系统中的空闲 UE 数量逐渐减少, 任务规模不断扩大, 系统环境变得更加复杂. 在此情况下, 采用完全分布式独立训练的 SA-Dep 难以有效应对复杂的 MEC 决策环境, 因而其性能逐渐劣于基于 CTDE 框架的 MA-Dep. 当任务请求概率进一步增大时, 系统中可用于协同计算的空闲 UE 显著减少甚至不存在, 两种算法均难以充分利用空闲 UE 资源进行协同计算, 从而导致性能差距再次缩小. 相比之下, GMA-Dep 能够在上述不同负载条件下利用 GAT 对系统状态与 UE 间交互关系进行高效建模, 从而在不同任务请求概率下均表现出最优的性能.

5.2.6　子任务数量对不同算法性能影响

为验证不同规模依赖型任务下所提算法性能, 图 8 给出了每个 UE 不同子任务数量对各算法性能影响. 实验中每种任务的子任务数量设置为 3~7 个, 其他参数均为 5.1 小节中的默认值. 从图 8 中可以观察到, 随着子任务数量的增加, 所有算法的系统时延呈现递增趋势. 这是由于任务的完成时间取决于退出任务的完成时间, 当构成该任务的子任务数量增加时, 意味着退出任务的前驱任务数量增加. 由于子任务之间执行的依赖性, 退出任务只能在其前驱任务计算完成后才能执行, 因此, 前驱任务数量的增加导致计算时延的增加, 进而导致退出任务完成时间增加, 即任务计算时延增加. 如图 8(b) 所示, 所有算法的能效随着子任务数量增加而降低. 这是由于在边缘节点服务资源受限的场景下, 子任务数量的增加导致更多的子任务在本地或空闲 UE 计算, 而计算资源有限的本地设备无法满足任务的计算需求, 从而导致任务完成率降低 (如图 8(d) 所示), 进而导致能效降低. 而根据公式 (28) 系统效用的定义, 如图 8(a) 所示, 系统效用也随着子任务数量的增加而下降.

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

此外, 观察图 8(b) 可以发现, 当子任务数量较大时 (6, 7), no-End2End 的能效高于其他算法. 这是由于 no-End2End 不考虑端端协同计算, 随着子任务数量的增加, 更多子任务被迫在本地设备上执行. 根据本地设备计算能耗公式 (7) 和约束条件 (C5), 在本地可用计算资源受限的情况下, 子任务数量的增加会导致分配给单个子任务的计算资源下降, 从而降低单个子任务的计算能耗. 结合式 (27) 中能效的定义, 在部分子任务计算失败但系统能耗显著降低的情况下, no-End2End 在子任务数量较大时反而表现出更高的能效. 然而, 该能效提升是以任务完成率和服务质量下降为代价的, 难以满足实际 MEC 场景下对可靠性和时延性能的要求. 总之, 与所有基准算法相比, 本文提出的 GMA-Dep 在不同子任务数量的场景下均展现出最优的系统效用、能效、时延和任务完成率, 验证了该算法能够针对依赖型任务作出更优的卸载策略.

6　结束语

本文针对资源受限场景下多样化依赖任务的计算卸载问题, 提出了一种分布式智能协同计算卸载和资源分配方法. 首先, 综合考虑边缘节点的服务能力限制以及任务的执行依赖关系, 构建了多用户多边缘节点 MEC 场景下的多节点协同计算卸载模型. 在此基础上, 通过优化任务调度顺序、计算卸载决策、传输功率分配决策和计算资源分配决策, 以最大化系统长期平均效用为目标, 建立了相应的优化问题, 并提出了 GMA-Dep 算法进行求解. 该算法为每个用户配置一个 GAT 增强的 DDPG 网络与子任务调度模块, 分别用于学习计算卸载及资源分配策略和确定依赖型子任务的调度顺序; 同时, 算法采用集中式训练与分布式执行架构, 在考虑全局优化的同时实现了分布式智能协同计算卸载. 为了验证所提方法的有效性, 本文在不同网络配置下进行了仿真实验, 包括用户设备数量、边缘节点数量、时延 – 能效权重系数、任务请求概率和子任务数量等多个维度. 实验结果表明, 所提 GMA-Dep 算法整体性能显著优于多种现有基线方法, 验证了所提算法能够有效协调用户设备和边缘节点的资源进行协同计算. 未来工作中, 将在现有研究基础上面向用户高速移动场景, 进一步研究依赖型任务的智能协同计算与卸载机制, 综合考虑带宽约束、子任务等待与启动等多维开销因素, 并联合计算卸载与服务缓存开展协同优化.

参考文献

1 Luo Q, Hu S, Li C, et al. Resource scheduling in edge computing: a survey. IEEE Commun Surv Tutorials, 2021, 23: 2131–2165
2 Gu H, Zhao L, Han Z, et al. AI-enhanced cloud-edge-terminal collaborative network: survey, applications, and future directions. IEEE Commun Surv Tutorials, 2024, 26: 1322–1385
3 Zhao W, Chellappa R, Phillips P J, et al. Face recognition. ACM Comput Surv, 2003, 35: 399–458
4 Liu F, Huang J, Wang X. Joint task offloading and resource allocation for device-edge-cloud collaboration with subtask dependencies. IEEE Trans Cloud Comput, 2023, 11: 3027–3039
5 Lou J, Tang Z Q, Jia W J, et al. Startup-aware dependent task scheduling with bandwidth constraints in edge computing. IEEE Trans Mobile Comput, 2024, 23: 1586–1600
6 Lou J, Tang Z, Zhang S, et al. Cost-effective scheduling for dependent tasks with tight deadline constraints in mobile edge computing. IEEE Trans Mobile Comput, 2023, 22: 5829–5845
7 Liu J, Zhang Y, Ren J, et al. Auction-based dependent task offloading for IoT users in edge clouds. IEEE Internet Things J, 2023, 10: 4907–4921
8 Chen X, Cao J, Sahni Y, et al. Dynamic task offloading in edge computing based on dependency-aware reinforcement learning. IEEE Trans Cloud Comput, 2024, 12: 594–608
9 Chen X, Cao J, Sahni Y, et al. Mobility-aware dependent task offloading in edge computing: a digital twin-assisted reinforcement learning approach. IEEE Trans Mobile Comput, 2025, 24: 2979–2994
10 Gao H, Wang X, Wei W, et al. Com-DDPG: task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the Internet of Vehicles. IEEE Trans Veh Technol, 2024, 73: 348–361
11 Fang J, Qu D Z, Chen H J, et al. Dependency-aware dynamic task offloading based on deep reinforcement learning in mobile edge computing. IEEE Trans Netw Sci Eng, 2024, 21: 1403–1415
12 Cao Z, Deng X, Yue S, et al. Dependent task offloading in edge computing using GNN and deep reinforcement learning. IEEE Internet Things J, 2024, 11: 21632–21646
13 Zhou X, Ge S, Liu P, et al. DAG-based dependent tasks offloading in MEC-enabled IoT with soft cooperation. IEEE Trans Mobile Comput, 2024, 23: 6908–6920
14 Xie B, Cui H, He Y, et al. A deep reinforcement learning approach for dependent task offloading in multi-access edge computing. IEEE Trans Cogn Commun Netw, 2025, 11: 2601–2617
15 Zeng J, Kuang Z, Chen R, et al. Delay-sensitive dependent tasks offloading and resource allocation in VEC: a deep-reinforcement learning approach. IEEE Internet Things J, 2025, 12: 34190–34203

https://www.sciengine.com/doi/10.1360/SSI-2025-0416

16 Liu S, Yu Y, Lian X, et al. Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. IEEE J Sel Areas Commun, 2023, 41: 538–554
17 Zhao G, Xu H, Zhao Y, et al. Offloading tasks with dependency and service caching in mobile edge computing. IEEE Trans Parallel Distrib Syst, 2021, 32: 2777–2792
18 Lv X Y, Du H W, and Q. Ye. TBTOA: a DAG-based task offloading scheme for mobile edge computing. In: Proceedings of IEEE Int Conf Commun, 2022. 4607–4612
19 Gong Y, Hao F, Wang L, et al. A socially-aware dependent tasks offloading strategy in mobile edge computing. IEEE Trans Sustain Comput, 2023, 8: 328–342
20 Zhang J, Zhang G, Bao X, et al. Dependent application offloading in edge computing. IEEE Trans Cloud Comput, 2023, 11: 3439–3451
21 Shen Q, Hu B J, Xia E. Dependency-aware task offloading and service caching in vehicular edge computing. IEEE Trans Veh Technol, 2022, 71: 13182–13197
22 Chen S, Rui L, Gao Z, et al. Cache-assisted collaborative task offloading and resource allocation strategy: a metareinforcement learning approach. IEEE Internet Things J, 2022, 9: 19823–19842
23 Liang Y, Zhang S, Wu J. Cost-efficient delay-bounded dependent task offloading with service caching at edges. IEEE Trans Comput, 2025, 74: 3568–3581
24 Guo H, Wang Y, Liu J, et al. Multi-UAV cooperative task offloading and resource allocation in 5G advanced and beyond. IEEE Trans Wireless Commun, 2024, 23: 347–359
25 Chen X, Hu S, Yu C, et al. Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning. IEEE Trans Parallel Distrib Syst, 2024, 35: 391–404
26 Arabnejad H, Barbosa J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans Parallel Distrib Syst, 2013, 25: 682–694

Intelligent collaborative computation offloading for dependent tasks in edge networks

Zhixiu YAO[1], Yun LI[1,2], Shichao XIA[1*], Yi JIANG[1] & Jishen LIANG[2,3]

1. *School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China*
2. *School of Communications and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China*
3. *Communications NCO Academy, Engineering University of the Information Support Force, Chongqing 400035, China*
* Corresponding author. E-mail: xiashichao@cqupt.edu.cn

Abstract The deep integration of artificial intelligence and mobile edge computing has become a key driving force for realizing the vision of endogenous intelligence in 6G networks, where intelligent computation offloading plays a crucial role and has attracted extensive attention. This paper investigates the computation offloading problem of diverse dependent tasks in resource-constrained edge computing environments and proposes a distributed intelligent collaborative computation offloading approach. Specifically, by jointly considering limited edge service capabilities, task heterogeneity, and execution dependencies, we formulate an end-edge collaborative computation offloading and multi-dimensional resource allocation optimization problem, with the objective of maximizing the long-term average task processing utility under service delay tolerance constraints. To solve the formulated problem, a graph attention-based multi-agent deep reinforcement learning algorithm for dependent task computation offloading is developed. During the centralized training phase, a graph attention network is employed to capture critical global state information and learn efficient collaborative offloading policies, while in the decentralized execution phase, each end user independently makes decisions based solely on local observations, thereby enabling distributed intelligent collaborative computation offloading. Finally, extensive simulation experiments are conducted to evaluate the performance of the proposed algorithm. The results demonstrate that, compared with existing approaches, the proposed method consistently achieves superior performance under various network parameter settings.

Keywords edge intelligence, computation offloading, dependent task, collaborative computing, multi-agent deep reinforcement learning

https://www.sciengine.com/doi/10.1360/SSI-2025-0416