车辆边缘计算中基于深度学习的任务判别卸载

章坚武[1]，戚可寒[2]，章谦骅[3*]，孙玲芬[4]

（1. 杭州电子科技大学 信息工程学院，杭州 310018；2. 杭州电子科技大学 通信工程学院，杭州 310018；
3. 浙江大学 信息与电子工程学院，杭州 311121；4. 普利茅斯大学，普利茅斯 PL48AA）

摘要 车辆边缘计算（VEC）将移动边缘计算（MEC）与车联网（IoV）技术相结合，将车载任务下沉至网络边缘，以此解决车辆终端计算能力有限问题。为了克服任务数量骤增的车载任务调度难题并提供一个低时延服务环境，首先依据所选的 5 大特征参数的动态关联变化准则，使用改进型层次分析法（AHP）将车载任务划分为 3 类主要任务，基于 3 种卸载决策进行资源分配联合建模；随后，利用调度算法和罚函数来消除建模的约束条件，所获的代价值为之后的深度学习算法提供输入；最后，提出一种基于深度学习的分布式卸载网络算法来有效降低 VEC 系统的能耗与时延。仿真实验结果表明，所提卸载方案相较传统深度学习卸载方案具有更好环境适应性与稳定性，并降低了任务平均处理时延与能耗。

关键词 深度学习；边缘卸载；多约束优化；任务类型划分；车辆边缘计算
中图分类号 TN929.5 **文献标志码** A DOI 10.12178/1001-0548.2022376

Deep Learning-Based Task Discrimination Offloading in Vehicular Edge Computing

ZHANG Jianwu[1], QI Kehan[2], ZHANG Qianhua[3*], and SUN Lingfen[4]

(1. College of Information Engineering, Hangzhou Dianzi University, Hangzhou 310018, China; 2. College of Communication Engineering, Hangzhou Dianzi University, Hangzhou 310018, China; 3. College of Information Science & Electronic Engineering, Zhejiang University, Hangzhou 311121, China; 4. University of Plymouth, Plymouth PL48AA, United Kingdom)

Abstract Vehicle Edge Computing (VEC), combining mobile edge computing (MEC) with the Internet of Vehicles (IoV) technology, offloads vehicle tasks to the edge of the network to solve the problem of limited computing power at the vehicle terminal. In order to overcome the difficulty of on-board task scheduling due to the sudden increase in the number of tasks and provide a low-latency service environment, the vehicle tasks are divided into three types of main tasks by using improved Analytic Hierarchy Process (AHP) according to the dynamic correlation change criteria of the selected five feature parameters, and the joint modeling of resource allocation is carried out based on three kinds of offloading decisions. Then, the constraints of the modeling are eliminated by using scheduling algorithm and penalty function, and the obtained substitution value is taken as the input for the following deep learning algorithm. Finally, a distributed offloading network based on deep learning is proposed to effectively reduce the energy consumption and delay of VEC system. The simulation results show that the proposed offloading scheme is more stable than traditional deep learning offloading scheme and has better environmental adaptability with its less average task processing delay and energy consumption.

Key words deep learning; edge offloading; multi-constraint optimization; task type division; vehicular edge computing

近年来，伴随着 5G 站上新的高峰，物联网（Internet of Things, IoT）和无线通信技术不断发展并完善。虚拟现实、超高清视频等各类新型移动应用[1-2]蜂拥而至，导致移动终端（Mobile Devices, MD）的数量呈爆炸性增长，移动资源的大量消耗。作为 5G 的关键技术，移动边缘计算（Mobile Edge Computing, MEC）被认为是解决计算任务需求不断增长[3]和用户自身计算能力及资源有限的一

收稿日期：2022−11−07；修回日期：2023−02−22
基金项目：国家自然科学基金国际合作交流项目（IEC\NSFC\181300）；浙江省自然科学基金重点项目（LZ23F010001）
作者简介：章坚武，博士，教授，主要从事通信与 AI 技术融合方面的研究。
*通信作者 E-mail: zhangqh@zhejianglab.com

种强大范例[4]。边缘卸载作为 MEC 的核心技术，可以将计算密集型与时延敏感型应用程序/计算任务下沉至相比远程云服务器更接近移动设备的边缘服务器，从而降低执行终端任务的时延与能耗，还可有效提高服务质量（Quality of Service, QoS）与用户体验质量（Quality of Experience, QoE）。

车联网（Internet of Vehicles, IoV）实现车与车、车与人、车与基础设备等车与一切（Vehicle to Everything, V2X）的信息交互[5]。5G 与 IoV 的协作[6] 可以提供一个安全、舒适、精准的交通环境，并且可以改善交通状况、减少交通事故[7] 与网络拥塞。

车辆边缘计算（Vehicular Edge Computing, VEC）作为 MEC 与 IoV 相结合的改革创新技术，目前受到了较为广泛的关注。VEC 通过连接 MEC 服务器以及路侧单元[8]（Road Side Unit, RSU）为车辆提供服务。RSU 作为部署在路侧的通信网关，是 IoV 中连接的核心。

为了有效缓解处理计算密集型任务的压力，通过提出不同低复杂度算法来确定计算卸载决策，权衡时延与能耗，降低整体 VEC 系统的代价消耗。文献 [9] 针对通信场景中 MEC 的部署提出了一种基于改进型粒子群算法（Particle Swarm Optimization, PSO）的计算卸载策略，有效降低了 MEC 服务器的处理时延并且平衡了其负载。文献 [10] 为了充分利用车辆 CPU 资源提出了反向卸载框架，利用分布式对偶分解法与基于高效搜索的贪婪算法（Greedy based on Efficient Searching, GES）来获得逆向卸载决策，从而降低时延。文献 [11] 将软件定义网络（Software Defined Network, SDN）技术引入传统网络体系，设计了一种由 SDN 控制的双层分布式 VEC 结构，并提出了一种动态服务卸载与迁移算法，以此提高区块链系统吞吐量，分别降低了执行时延与能耗。文献 [12] 通过有向无环图（Directed Acyclic Graph, DAG）将任务序列切割成基本单元以此得到简化的最优卸载策略。文献 [13] 考虑有效经济地利用志愿者辅助车辆的空闲计算资源来帮助过载 MEC 服务器，并基于 Stackelberg 博弈与遗传算法确定最优 VEC 卸载策略，最大化志愿车辆奖励并降低车辆卸载成本。文献 [14] 结合 IoT 设备、MEC 服务器与 MCC 服务器实现了一种能量高效利用的动态任务卸载算法，同时结合李雅普诺夫算法，确定了最优计算位置。文献 [15] 利用停车场车辆作为虚拟 MEC 服务器来进行辅助计算，建立了基于随机森林的时间相关轨迹预测模型。此模型能够很好应对任务请求量大的情况，并可以实现结果的准确反馈。

深度学习使用具有多个处理层的深度神经网络（Deep Neural Networks, DNN）来学习数据，将其应用于 IoV 环境中，增强环境感知、突破处理卸载决策与资源分配问题时的维度限制已成为当今 VEC 研究的热门方向。文献 [16] 建立 DNN 延迟预测模型来表征已被划分的各个子任务的计算时延，并设计了插槽模型来调度子任务，将深度学习应用于强化学习，可使 VEC 系统拥有更好的环境感知能力。文献 [17] 通过马尔可夫决策（Markov Decision Process, MDP）建模，利用深度强化学习（Deep Reinforcement Learning, DRL）来处理高速行驶车辆的任务调度空间状态。文献 [18] 研究了车联网下应用程序的子任务拓扑，利用 DAG 获取任务优先级并提出了一种基于策略梯度的分布式 DRL，相比现有方案拥有更好的卸载性能。文献 [19] 提出了一种基于联邦学习的 MEC 隐私保护方法，能够实现高准确率内容保护，适用于环境需求较高的边缘计算场景。文献 [20] 考虑了 VEC 环境的内在特性，提出了一种基于极度强化学习（Extreme Reinforcement Learning, ERL）的上下感知 VEC 任务调度程序，实现了 96% 以上的 VEC 任务完成率。文献 [21] 提出了一种视觉结合 DRL 的自适应视频流方法。该方法在确保卸载数据质量的同时，减少了系统的能耗与时延。

目前工作中，划分车载任务并发布针对性卸载决策的研究较少。最大处理容忍时延、车辆的剩余电量、路段的任务总量均有可能动态地影响决策制定。为了能够节约 VEC 系统资源并且保障用户的服务体验，本文提出了一种多移动终端、单 MEC 服务器以及云服务器所组成的 VEC 网络卸载架构。

1 模型建立

1.1 网络模型

本文系统模型如图 1 所示，根据不同地理信息在各路段配备 RSU，车辆可以通过 V2I 通信方式与所在区域的 RSU 进行通信。为了便于描述，在单 RSU 所覆盖的区域中展开系统模型。考虑发起请求车辆可以在本地进行任务计算，也可以卸载任务至该路段配备 MEC 服务器的 RSU 上进行计算。车辆在 RSU 范围内存在多计算任务，定义 N 台车

辆组成的车辆用户集合为$U=\{U_1,U_2,\cdots,U_n\}$。每辆车均拥有一个或多个相互独立的任务需要执行，定义车辆 n 的任务集合为$M=\{1,2,\cdots,m\}$。除去用于处理计算的 MEC 服务器以外，RSU 还配备一个前置服务器用作任务判别与决策生成。前置服务器将车载任务划分为能耗敏感型、时延敏感型以及非敏感型 3 类，并依据 MEC 服务器剩余资源向各车辆发布卸载决策，卸载决策用一个二进制变量$b_{nm}\in\{0,1\}$表示。b_{nm}=0 表示车辆 n 将在本地处理任务 m；b_{nm}=1 表示车辆 n 将其任务 m 卸载到 MEC 服务器，各车辆将依据决策在本地有序执行任务并将需卸载任务上传至 MEC 服务器缓存。

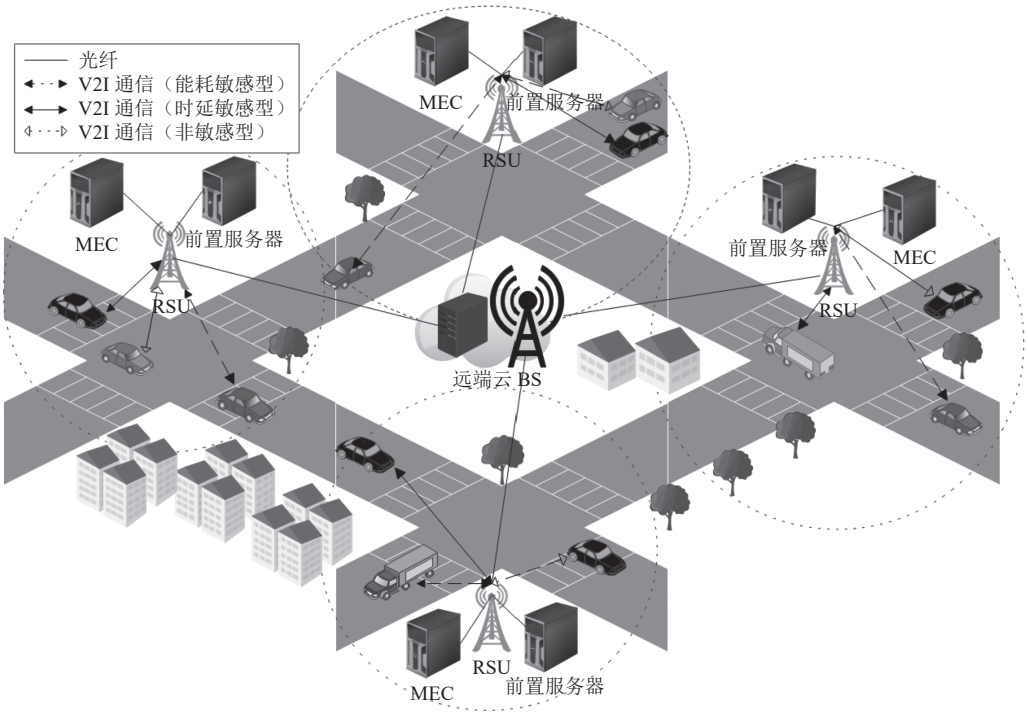


图 1　VEC 网络模型

同一时刻 RSU 接收到的任务将按照发起顺序进入服务器缓存等待处理，当前 MEC 服务器存在最大计算资源。分配给缓存中待处理任务相应的服务器处理资源，需要考虑到之前未完成任务所占用的资源。同时，若待处理任务所需计算资源超越服务器剩余计算资源，则会导致任务处理失败。因此，考虑将该任务由 MEC 服务器转发至远端云服务器，通过服务器的资源调度来解决任务：

$$d=\begin{cases}0 & f_{nm}^{c}\leqslant f_{\text{residue}}^{c}\\ 1 & f_{nm}^{c}>f_{\text{residue}}^{c}\end{cases}\tag{1}$$

式中，当 d=1 时，车辆 n 的任务 m 所需计算资源f_{nm}^{c}大于当前服务器空闲计算资源f_{residue}^{c}，因此该任务需要转发至云服务器进行处理；d=0 时，该任务不需要转发至云服务器。

1.2　任务判别模型

本文使用改进型层次分析法（Analytic Hierarchy Process, AHP）将车载任务划分为时延敏感、能耗敏感与非敏感 3 种类型。首先，为了构建判断矩阵，选择任务最大处理时延$T_{nm}^{\max}$、路段任务密集度、车载 CPU 剩余电量 Z_n、任务数据量以及车载 CPU 缓存性能这 5 条指标作为模型的判别准则层。其中，任务最大处理时延是主要判别因素；其余准则均为次要判别因素。每一个判别准则将进行层次单排序并分别进行一致性检验，因此可以得到车辆 n 任务 m 的一致判别矩阵$\boldsymbol{K}_{g\times g}=(k_{ij})_{g\times g}$：

$$k_{ij}=\begin{cases}\dfrac{1}{k_{ji}}\in\{1,2,\cdots,9\} & i\neq j\\ 1 & i=j\end{cases}\tag{2}$$

式中，g 为判别准则的个数；k_{ij} 为准则 i 对准则 j 的重要性判别。每个判别准则将在不同车载任务中被分配一个权重：

$$w_{nm}^{i}=\frac{\sqrt[g]{\prod_{j=1}^{g}k_{ij}}}{\sum_{i=1}^{g}\sqrt[g]{\prod_{j=1}^{g}k_{ij}}}\tag{3}$$

将同一时刻 RSU 接收到的所有任务的权重对应向量构造成准则权重向量矩阵：

$$\boldsymbol{\Delta}_1=\begin{bmatrix} w_{11}^1 & w_{11}^2 & w_{11}^3 & w_{11}^4 & w_{11}^5 \\ w_{12}^1 & w_{12}^2 & w_{12}^3 & w_{12}^4 & w_{12}^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{1m}^1 & w_{1m}^2 & w_{1m}^3 & w_{1m}^4 & w_{1m}^5 \\ w_{21}^1 & w_{21}^2 & w_{21}^3 & w_{21}^4 & w_{21}^5 \\ w_{22}^1 & w_{22}^2 & w_{22}^3 & w_{22}^4 & w_{22}^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{2m}^1 & w_{2m}^2 & w_{2m}^3 & w_{2m}^4 & w_{2m}^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{n1}^1 & w_{n1}^2 & w_{n1}^3 & w_{n1}^4 & w_{n1}^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{nm}^1 & w_{nm}^2 & w_{nm}^3 & w_{nm}^4 & w_{nm}^5 \end{bmatrix}_{nm\times 5} \tag{4}$$

判定任务类型为该模型的目标层，需要对上述得到的准则层次单排序进行总排序。针对需要减少处理时延的情况，最大处理时延、任务数据量等准则的依赖性提高；针对需要降低 CPU 损耗的情况，剩余电量以及 CPU 缓存的依赖性提高。每一个判别目标将进行层次总排序并分别进行一致性检验，因此可以构造目标权重向量矩阵：

$$\boldsymbol{\Delta}_2=\begin{bmatrix} \varphi_t^1 & \varphi_s^1 & \varphi_e^1 \\ \varphi_t^2 & \varphi_s^2 & \varphi_e^2 \\ \vdots & \vdots & \vdots \\ \varphi_t^5 & \varphi_s^5 & \varphi_e^5 \end{bmatrix}_{5\times 3} \tag{5}$$

式中，φ_t^i、φ_s^i以及φ_e^i表示划分成的 3 种目标对于 i 种准则的权重占比。最终，任务划分为 3 种类型，各自的表现分可由矩阵$\boldsymbol{\Delta}_{3(nm\times 3)}=\boldsymbol{\Delta}_1\times\boldsymbol{\Delta}_2$表示：

$$\boldsymbol{\Delta}_3=\begin{bmatrix} y_{11}^t & y_{11}^s & y_{11}^e \\ y_{12}^t & y_{12}^s & y_{12}^e \\ \vdots & \vdots & \vdots \\ y_{nm}^t & y_{nm}^s & y_{nm}^e \end{bmatrix}_{nm\times 3} \tag{6}$$

式中，y_{nm}^t、y_{nm}^s以及y_{nm}^e表示车辆 n 任务 m 被划分为时延敏感型、非敏感型以及能耗敏感型各自的表现分。选取最大表现$y_{nm}=\max\{y_{nm}^t,y_{nm}^s,y_{nm}^e\}$所对应的类型作为任务最终的判别。

1.3 通信模型

本模型中，RSU 半径覆盖范围内多辆车同时卸载任务，考虑了 OFDMA 系统，因此可以忽略不同独立车辆的各项任务之间的传输干扰。与此同时，假设带宽平均分配给各连接的车辆，于是可以得到车辆 n 第 m 个任务的上行传输速率[18]：

$$R_{nm}=\overline{B}\log_2\left(1+\frac{P_{nm}^{\mathrm{up}}H_n}{N_0}\right) \tag{7}$$

式中，$\bar{B}$表示车辆 n 上行链路的子信道带宽，$\bar{B}=B/n$，B 代表上行链路总带宽；H_n 表示车辆 n 与 RSU 之间的无线信道增益；N_0 为高斯白噪声功率，$N_0=\sigma^2$，σ 表示信道环境中高斯白噪声的标准差；P_{nm}^{up}表示针对任务 m 车辆 n 的发射功率。

1.4 计算模型

1.4.1 本地计算

针对车辆 n 决定利用本地 CPU 执行任务的情况进行建模。首先，定义车辆本身 CPU 的计算能力为f_n^{local}。由于任务仅在本地进行计算，不用考虑任务的传输时延，因此车辆 n 在本地执行任务 m 所消耗的时延为：

$$T_{nm}^{\mathrm{local}}=\frac{a_{nm}}{f_n^{\mathrm{local}}} \tag{8}$$

式中，a_{nm} 表示车辆 n 第 m 个任务的工作量大小。在给定具体的卸载决策$\{b_{nm}\}$后，可以得到车辆 n 本地计算所消耗的总时延为：

$$T_n^{\mathrm{local}}=\sum_{m=1}^{M}(1-b_{nm})T_{nm}^{\mathrm{local}} \tag{9}$$

定义e_n^{local}为车辆 n 在本地执行每 1 bit 数据的局部能耗，因此本地执行任务所消耗的总能量为：

$$E_{nm}^{\mathrm{local}}=a_{nm}e_n^{\mathrm{local}} \tag{10}$$

1.4.2 边缘计算

针对车辆 n 决定将任务卸载至边缘服务器的情况进行建模。考虑 MEC 服务器直接配置在 RSU 上这一种情况，因此忽略由 RSU 将任务转发至 MEC 服务器的能耗与时延。一般情况下认为边缘服务器具有足够的电源，因此车辆 n 将任务 m 卸载到边缘服务器的传输时延为：

$$T_{nm}^t=\frac{a_{nm}}{R_{nm}} \tag{11}$$

与此同时，任务在边缘服务器上进行处理的计算时延为：

$$T_{nm}^{\mathrm{off}}=\frac{a_{nm}}{f_{nm}^c} \tag{12}$$

式中，f_{nm}^c为边缘服务器分配给该任务的处理资源。由于 MEC 服务器反馈给车辆 n 计算结果的数

据大小相较于输入数据很小，因此本文不考虑下行传输以及车辆 n 接收处理结果的时延与能耗。在给定具体的卸载决策$\{b_{nm}\}$之后，可以得到车辆 n 卸载总时延：

$$T_n^{\text{off}} = \sum_{m=1}^{M} b_{nm}(T_{nm}^t + T_{nm}^{\text{off}}) \tag{13}$$

车辆 n 传输任务 m 所消耗的能量为：

$$E_{nm}^t = P_{nm}^{\text{up}} \frac{a_{nm}}{R_{nm}} \tag{14}$$

因此，可以得到车辆 n 将其任务 m 卸载至边缘服务器的总能耗为：

$$E_{nm}^{\text{off}} = E_{nm}^t + \frac{\mu P_{\text{mec}} a_{nm}}{f_{nm}^c} \tag{15}$$

式中，定义 P_{mec} 为 MEC 服务器的设备功率；μ为 MEC 服务器的能耗占比。当μ=0 时，整个卸载过程只考虑车辆 n 所消耗的能量。

1.4.3　云计算

当缓存队列中即将被处理的任务所需计算资源超过 MEC 空闲计算资源，该任务将从 MEC 服务器转送至 BS 进而转发至远端云服务器。由于车辆已执行卸载过程，且转发过程均借助光纤，则不需考虑上下行传输时延与能耗。因此车辆 n 的任务 m 在云服务器上的计算时延为：

$$T_{nm}^{\text{cloud}} = \frac{a_{nm}}{f^{\text{cloud}}} \tag{16}$$

式中，f^{cloud} 为云服务器提供的计算资源。定义 P_{cloud} 为云服务器的设备功率，则车辆 n 的任务 m 在云服务器上的计算能耗为：

$$E_{nm}^{\text{cloud}} = \frac{\lambda P_{\text{cloud}} a_{nm}}{f^{\text{cloud}}} \tag{17}$$

式中，λ 为云服务器执行任务的能耗占比，当 λ=0 时，针对于整个卸载过程只考虑车辆 n 所消耗的能量。云计算模型建立在边缘计算模型基础之上，在给定具体的卸载决策$\{b_{nm}\}$之后，b_{nm}=1 的任务会有两种卸载方式，因此改写式 (13) 和式 (15)，得到新的车辆 n 卸载总时延与车辆 n 卸载任务 m 的总能耗分别为：

$$T_n^{\text{off}*} = \sum_{m=1}^{M} b_{nm}(T_{nm}^t + (1-d)T_{nm}^c + dT_{nm}^{\text{cloud}}) \tag{18}$$

$$E_{nm}^{\text{off}*} = E_{nm}^t + \frac{(1-d)\mu P_{\text{mec}} a_{nm}}{f_{nm}^c} + \frac{d\lambda P_{\text{cloud}} a_{nm}}{f^{\text{cloud}}} \tag{19}$$

式中，当 d=1 时，该任务需要转发至云服务器；d=0 时，该任务不需要转发至云服务器。

1.5　问题建模

本文寻求最小化所有车辆任务完成的总时延与总能耗之和。基于上述 3 种场景，将车辆 n 第 m 个任务的代价定义为能耗与时延的权衡值，引入任务的最大表现分作为权衡值的系数，并对不同类型的任务求取相应的最终权衡，得：

$$C_{nm} = \begin{cases} (1-y_{nm})[E_{nm}^{\text{local}}(1-b_{nm}) + E_{nm}^{\text{off}*} b_{nm}] + \\ y_{nm} \max\{T_n^{\text{off}*}, T_n^{\text{local}}\} \\ y_{nm} = y_{nm}^t \\ y_{nm}[E_{nm}^{\text{local}}(1-b_{nm}) + E_{nm}^{\text{off}*} b_{nm}] + \\ \max\{T_n^{\text{off}*}, T_n^{\text{local}}\}] \\ y_{nm} = y_{nm}^s \\ y_{nm}[E_{nm}^{\text{local}}(1-b_{nm}) + E_{nm}^{\text{off}*} b_{nm}] + \\ (1-y_{nm}) \max\{T_n^{\text{off}*}, T_n^{\text{local}}\} \\ y_{nm} = y_{nm}^e \end{cases} \tag{20}$$

式中，$T_n^{\text{off}*}$和$E_{nm}^{\text{off}*}$为式 (18) 和式 (19) 的计算结果，即加入云服务器辅助后的新时延与能耗。引入系统代价 C，其等效于执行所有任务的能耗与时延总加权值，以此来评估 VEC 系统的总体时延与能耗：

$$C = \sum_{n=1}^{N} \left(\sum_{m=1}^{M} C_{nm} \right) \tag{21}$$

在多个约束条件下，系统总代价 C 能够达到最小值。因此，系统的任务处理建模为：

$$\begin{aligned} &\min\ C \\ \text{s.t.}\quad &\text{C1}:\ \max\{T_n^{\text{local}}, T_n^{\text{off}*}\} \leqslant T_n^{\max} \\ &\text{C2}:\ b_{nm} \in \{0,1\} \\ &\text{C3}:\ \sum_{n=1}^{N}\sum_{m=1}^{M} f_{nm}^c \leqslant f_{\max}^c \\ &\text{C4}:\ y_{nm} \in (0,1) \\ &\text{C5}:\ n \in N, m \in M \end{aligned} \tag{22}$$

式中，C1 给出了车辆 n 执行所有任务的最大容忍时延；C2 给出了一个二进制的卸载决策；C3 限定分配给缓存队列中的任务计算资源不能超过 MEC 服务器的最大计算资源；C4 给出了各任务的判别；C5 给定了该系统下的车辆以及任务数量。

2　算法设计

本节提出了一种基于 DNN 的并行式 VEC 网络卸载架构，如图 2 所示。整体架构由独立并行的 K 个卸载单元、调度单元、罚函数以及一个有限大小的内存单元 4 部分组成。并行分布的卸载单

元以及调度单元用以实现算法中的决策生成部分，其中，每个卸载单元包含一个结构完全相同的 DNN；罚函数用以实现对约束条件的处理；内存单元用以辅助算法中学习与训练。该卸载架构将配备至各路段的前置服务器用以实现决策生成。

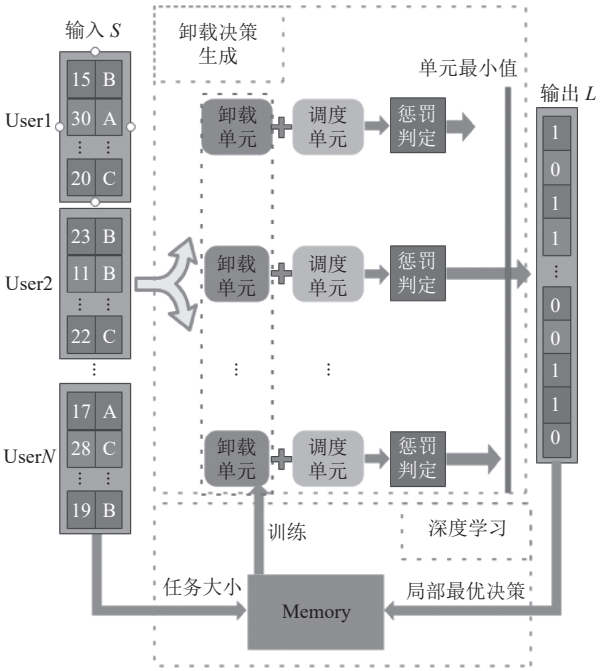


图 2 基于深度学习的 VEC 任务卸载网络架构

给定所有用户的输入数据大小，并将输入定义为 S。除存放输入数据的缓存以外，再配置一组缓存用以存放各项任务判别标签 Y={A,B,C}。其中，标签 A 对应$y_{nm}=y_{nm}^{t}$；标签 B 对应$y_{nm}=y_{nm}^{e}$；标签 C 对应$y_{nm}=y_{nm}^{s}$。

用户数量以及任务数量的增加会导致目标二元卸载决策序列{b}的大小呈指数型增长，在各条件制约下，生成最优卸载决策问题将变成一个 NP 难（NP-hard problem）问题。考虑使用调度算法以及罚函数减少约束并获取局部最优决策 L，将非凸问题转变成凸问题，再通过深度学习来解决。

2.1 局部最优决策

2.1.1 卸载单元与调度单元

针对每一组输入 S，需要找到一个合适的卸载策略函数。具体来说，使用 k 个卸载单元生成 k 个二进制卸载决策 b_k，如图 3 所示。每一个卸载单元均包含一个 DNN，$N\times M$ 个带有标签的任务作为网络的输入层，卸载决策 b_k 作为输出层。

置于卸载单元内的 k 个 DNN 具有相同的结构与不同的网络参数$\varphi_k\{w_k, v_k\}$。其中，线性关系系数 w_k 由正态分布初始化器 tf.random_normal_initializer 定义初值，偏倚系数 v_k 由常量初始化器 tf.constant_initializer 定义初值。带有标签的任务通过神经网络得到 k 个二进制决策，卸载策略函数的参数形式表现为：

$$f_{\varphi_k}: S \to b_k \tag{23}$$

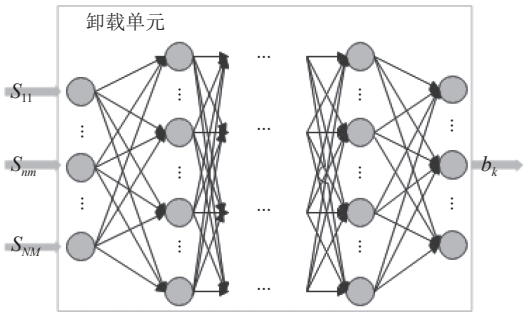


图 3 卸载单元

一旦得到 k 个二进制卸载决策，原始问题 P1 中的约束条件 C2、C4、C5 均可消除，原问题将转换为一个资源分配与容忍时延约束问题 (P2)：

$$\begin{aligned}&(\mathrm{P2}): \min_{S,b_k} C(S,b_k)\\ &\text{s.t.}\quad \mathrm{C1}: \max\{T_n^{\mathrm{local}}, T_n^{\mathrm{off}}\} \leqslant T_n^{\max}\\ &\mathrm{C3}: \sum_{n=1}^{N}\sum_{m=1}^{M} f_{nm}^{c} \leqslant f_{\max}^{c}\end{aligned} \tag{24}$$

由于分配给卸载任务的计算资源不能超过 MEC 服务器空闲计算资源，如图 4 所示，本文提出了一种调度算法。每一个缓存内的任务在处理之前访问当下空闲计算资源，若空闲资源不足以处理，任务将调度给云服务器，借助云服务器计算资源辅助处理任务。

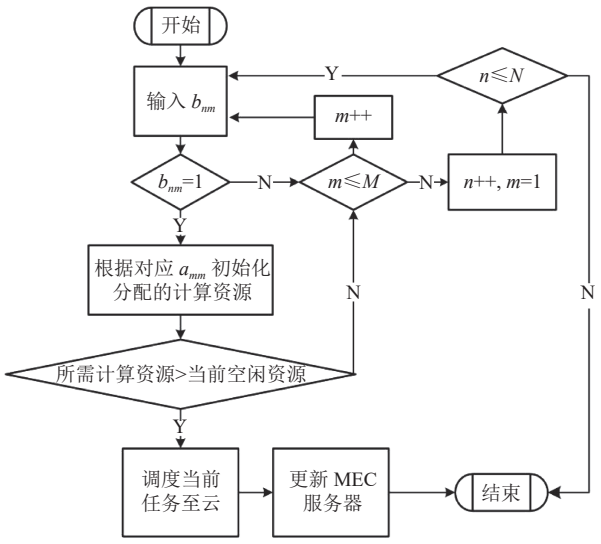


图 4 调度算法流程图

合理分配卸载任务至不同服务器，便可以更新代价 $C^*(S,b_k)$。解决资源分配问题后，仅剩约束条件 C1，因此 P2 将转换为一个最大容忍时延约束问题 (P3)：

$$(P3):\min_{S,b_k} C^*(S,b_k)$$
$$\text{s.t.}\quad C1:\max\{T_n^{\text{local}},T_n^{\text{off}}\}\leqslant T_n^{\max} \tag{25}$$

2.1.2　基于罚函数构造适应度函数

对于 P3 中仅存的不等式约束条件 C1，构造罚函数来彻底消除目标函数的约束。为了能够贴合所建模型的精度，选取两个函数来组成罚函数，并与目标函数联合构成最终的适应度函数。

首先，将约束条件 C1 进行移项：

$$g=\max\{T_n^{\text{local}},T_n^{\text{off}}\}-T_n^{\max}\leqslant 0 \tag{26}$$

其次，构造函数判断用户 n 任务执行时延是否超过最大容忍时延：

$$h=\max\{0,g\} \tag{27}$$

当 $h\leqslant 0$ 时，不需要惩罚目标函数；当 $h>0$ 时，需要惩罚目标函数。考虑到 RSU 覆盖范围有限，相对较长的时延不仅会降低用户体验，同样也会造成任务的执行失败。因此，设定超过最大容忍时延将对于系统代价造成指数型影响，从而得到惩罚函数：

$$\theta(h)=\begin{cases}0 & h\leqslant 0\\ 10h & 0<h<1\\ 5\mathrm{e}^h & 1\leqslant h<3\\ 7\mathrm{e}^h & 3\leqslant h<10\\ h\mathrm{e}^h & \text{otherwise}\end{cases} \tag{28}$$

适应度函数基于上述模型以及罚函数的建立，可以在最大容忍时延的约束下，计算整个系统的总代价。所得适应度函数值越大，则系统所消耗代价越大，反之则越小。目标函数与罚函数相加即可得到适应度函数：

$$\text{Fitness}\,(h)=\text{Fitness}(S,b_k)=C^*(S,b_k)+\theta(h) \tag{29}$$

在解决最大容忍时延问题 (P3) 后，本文在 k 组经过卸载单元、调度单元以及惩罚判定的候选对象中选取系统代价最小的卸载决策作为局部最优决策进行输出，这一组局部最优决策将与对应的输入 S 合并作为神经网络的训练标准。得到局部最优决策的过程可以表示为：

$$L=\arg\min\text{Fitness}(S,b_k) \tag{30}$$

2.2　算法流程与深度学习

VEC 中基于深度学习的任务卸载算法如下。

算法 1　VEC 中基于深度学习的任务卸载算法

Input: t 时刻得到的用户任务输入 $S_t(a_t,y_t)$

Output: 对应 t 时刻得到的局部最优卸载决策 L_t

1）使用随机权重初始化 k 个 DNN 网络参数 $\varphi_k\{w_k,v_k\}$

2）初始化标准库的大小 standard_size 并清空内存

3）for each episode do

4）for t=1 to T do

5）输入 S_t 通过 k 个卸载单元得到 k 组初步 VEC 代价 C_1 与卸载决策 $b_k=f_\varphi(k,S_t)$

6）if MEC 服务器需提供计算资源>$f_{\max}^c$

7）将超越项任务卸载至云服务器，调度单元重新计算代价 C^*

8）else $C^*=C_1$

9）end if

10）根据决策 b_k 判断任务执行时延是否超过车辆最大容忍时延

11）计算惩罚并得到最终 k 组 Fitness(S,b_k)

12）选择最小值对应的决策作为局部最优决策 $L=\arg\min\text{Fitness}(S,b_k)$

13）将 stan=(a_t,y_t,b_k) 作为准则存储到标准库 Standard

14）在 Standard 中随机抽选 k 组 stan 训练 k 个同结构 DNN 并更新网络参数φ_k

15）end for

16）end for

按照上述步骤，首先初始化 k 个 DNN 的网络参数并清空标准库内存。在每一个片段中，设置各组发起请求的任务作为神经网络不同时刻 t 的输入，具体操作如下：将同一时刻 RSU 接收到的任务请求分别放入 k 个 DNN 中，求代价值 C_1 与决策 b_k。考虑到每一个 RSU 仅配备一个 MEC 服务器，路段的高任务请求会导致需要的计算资源超过 MEC 最大计算资源。为了缓解这一问题，设置调度单元将任务转发至云服务器，并重新求取代价值 C^*。在结束步骤 9）以后，优化问题从 P1 演化成为 P2。考虑到 RSU 覆盖范围有限，设置$T_n^{\max}$作为车辆 n 在该覆盖面下的最大容忍时延。为了提高最大容忍时延的影响，提出一种指数型罚函数并由此构造适应度函数 Fitness 来权衡约束下的 VEC 总代价，并通过 argmin 函数得到局部最优卸载决策 L。将 (S_t,B) 展平作为新的数据内容存入标准库 Standard 中，作为 DNN 的训练准则。在完成一次

内容存放后，随机抽样 k 组训练准则分配给所有 DNN，利用梯度下降算法，最小化损失函数 loss 从而更新 DNN 网络参数，具体过程如下：

$$y_{\text{pred}} = \text{sigmoid}(\varphi_k\{w_k, v_k\}) = \frac{1}{1+\text{e}^{-\varphi_k}}$$
$$\text{loss} = -x_{\text{true}}\log(y_{\text{pred}}) - (1-x_{\text{true}})\log(1-\log(y_{\text{pred}})) \tag{31}$$

式中，x_{true} 是每一组输入 S 对应的最小代价卸载决策。当标准库 Standard 内存占满时，自动舍弃最原始数据并添加新的数据作为补充。

算法 1 给出的 VEC 中基于深度学习的任务卸载算法整体均使用 Tensorflow 实现，此算法被期望能够使神经网络通过训练，不断收敛直至生成最优卸载决策，并能够在极短时间内对测试任务队列做出决策。

3 仿真结果与分析

3.1 参数设置

本实验仿真环境均使用 Python3.8.6 和 Tensorflow2.6.0 实现。设置车辆数量 N=3，任务数量 M=3，每一个任务大小在 [10, 30] Mb 中随机取值。在通信模型中，车辆的发送功率P_{nm}^{up}=1 500 mW，信道带宽 B=1 MHz[12]，信道增益 h=2×10^{-4}，高斯白噪声功率 σ^2=6×10^{-14} W。本地计算模型中，车辆本地 CPU 计算能力f_n^{local}=2.105×10^6 bit/s，本地局部能耗系数e_n^{local}=3.25×10^{-7} J/bit。边缘计算模型中，服务器 CPU 计算速率为 2.5×10^{10} cycles/s，完成每 bit 任务所需周期数为 10^3 cycles/bit，MEC 服务器的计算能力f_{nm}^{c}=2.5×10^7 bit/s，MEC 服务器的设备功率 P_{mec}=25 W。云计算模型中，云服务器的计算能力f^{cloud}=5×10^7 bit/s，云服务器的设备功率 P_{cloud}=50 W。深度学习算法的学习率为 0.01，学习间隔为 10，标准库大小为 1 024，批大小为 128。

3.2 结果分析

本实验中，在相同 VEC 环境下选取全本地执行、全 MEC 卸载、随机卸载以及传统基于深度学习的分布式卸载[22]（Distributed Deep Learning-Based Offloading, DDLO）这 4 种方法与所提出的卸载方案进行比较。

1）全本地执行：将车联网服务需求全部配置在车辆本地进行处理，模拟网络拥塞或信号覆盖差的路段。

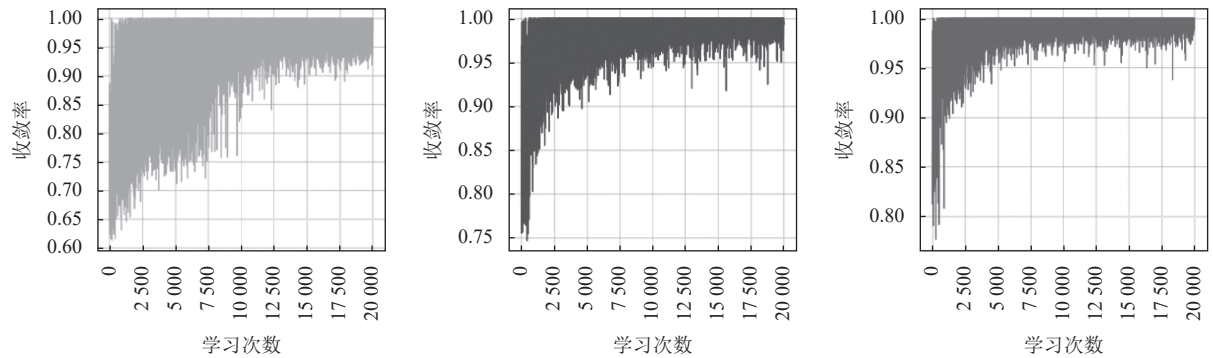
2）全 MEC 卸载：所有任务均通过边缘服务器进行处理，不考虑服务器缓存性能。

3）随机卸载：随机给用户分配计算资源进行卸载操作，模拟不提供卸载算法的服务方式。

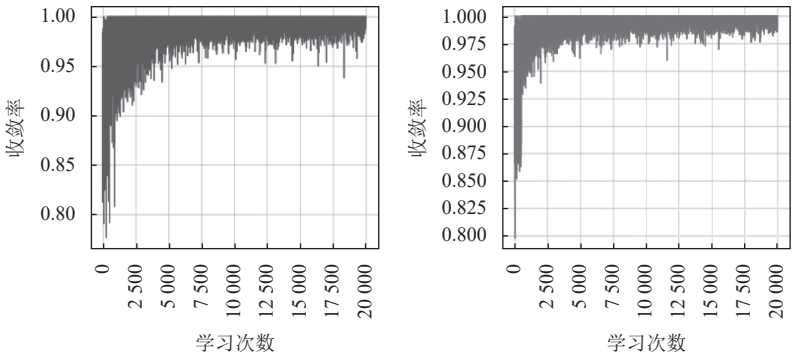
4）DDLO：一种基于深度学习的分布或卸载方法[22]。

3.2.1 收敛性能

图 5 显示了不同 DNN 个数下本文算法的收敛性能。随着输入 S 不断更新，枚举 2^{NM} 种不同的卸载决策，选取其中最小代价值作为评判基准，收敛率为本章算法输出的局部最优代价 Fitness(S,b_k) 与最小代价的比值。随着学习次数的增加，本算法逐渐收敛于 1，而 DNN 个数增多更会加快收敛的速度。由于需要通过比较选出局部最优卸载决策 L，因此 DNN 个数一定大于等于 2。当 DNN 个数为 2 时，学习次数在 13 000 次才能使收敛率达到 0.98；而当个数为 10 时，学习次数仅需 1 000 次便能使收敛率达到 0.98。过少个数的 DNN 会导致算法学习效率较低，而过多 DNN 会导致算法收敛过快，使得局部最优解出现的可能性大于全局最优解的可能性。因此在之后的仿真实验中，考虑到算法收敛能够比较平稳，均选取 DNN 个数为 4。



a. 2 个 DNN　b. 3 个 DNN　c. 4 个 DNN



d. 5 个 DNN　　e. 10 个 DNN

图 5　不同 DNN 个数下的收敛性能

3.2.2　系统效用

5 种算法在不同任务数量下的表现如图 6 所示。实验结果表明，随着任务数量增加，所有算法的平均系统代价均不断攀升。相比其余 4 种算法，本文提出的算法有 0.53%～44% 的性能提升。针对全本地执行算法而言，较低的 CPU 处理能力将带来巨大的处理时延以及能耗，因此系统代价一直居高不下。对于全卸载算法，当任务数量较少时，该算法比较优异。一旦任务密度增加，由于 MEC 服务器的计算能力存在阈值，其负载也会逐渐增大，因此系统代价会不断增大。而对于随机卸载算法，卸载决策一直没有办法达到最优，因此效果并不理想。本文提出的算法相较 DDLO 算法在任务数量骤增之后，有着良好且稳定的系统代价表现。

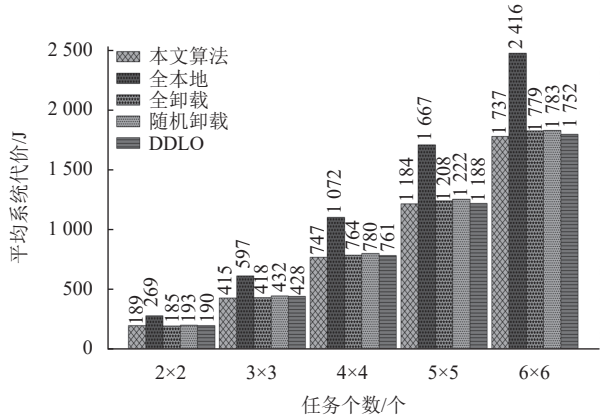


图 6　不同任务个数下的系统代价

图 7 给出了不同能耗权重μ和 λ 对 5 种算法的影响情况，该情况默认 3 种类型任务占比相同。本文提出了一种由 MEC 服务器转发超额任务至云服务器的概念，因此同时考虑两个服务器的能耗占比，将μ+λ 的和值作为影响因子。当影响因子从 0.5 上升至 3 时，相比其余 4 种算法，本文算法较大幅度降低了系统的平均代价。图 7 显示μ+λ 的值为 3 时，全本地、全卸载、随机卸载以及 DDLO 算法所生成的平均系统代价分别为 598 J、493 J、491 J、483 J，而使用本文算法，平均系统代价仅为 458 J，获得了 5.5%～30.6% 的性能提升。能耗权重的提升，不改变全本地算法的性能，但会对全卸载算法的系统代价造成巨大的影响。

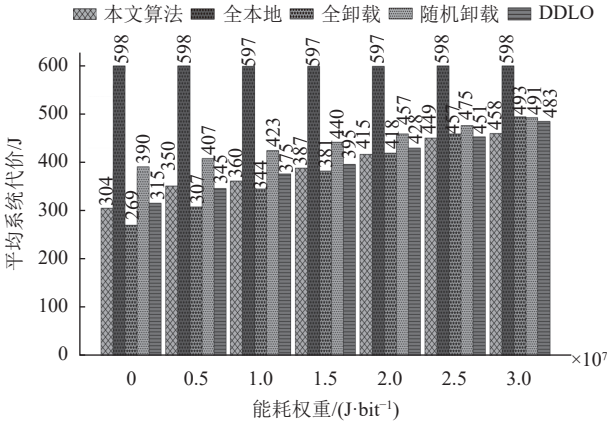


图 7　不同能耗权重下的系统代价

图 8 给出了能耗敏感型任务占比对 5 种算法的影响情况。当同一时刻发起请求的能耗敏感型任务占比增加，随机卸载算法的平均系统代价变化较为稳定，因为卸载决策随机生成，不会使系统有能耗或时延的偏重。全本地执行算法的系统代价不断下降，原因是本地执行不需要考虑信道传输、服务器执行的时延与能耗。对于时延敏感型任务占比较大的情况，由于 DDLO 算法不考虑任务类型的划分，卸载决策不具备针对性，因此造成系统代价较高。本文算法对比其余 4 种算法，在任务类型占比转变的时候平均系统代价值较小。当时延敏感型任务或能耗敏感型任务占比为 60% 时，采用本文算法能够降低 1.28%～56.8% 或 0.79%～33.4% 的平

均系统代价；当时延与能耗敏感型任务占比相同时，本文算法能够分别降低 43.8%、0.8%、4%、13.9% 的平均系统代价。因此，该实验结果证明了本文算法在时延敏感型任务占比较大时，能够更好地降低系统的代价。

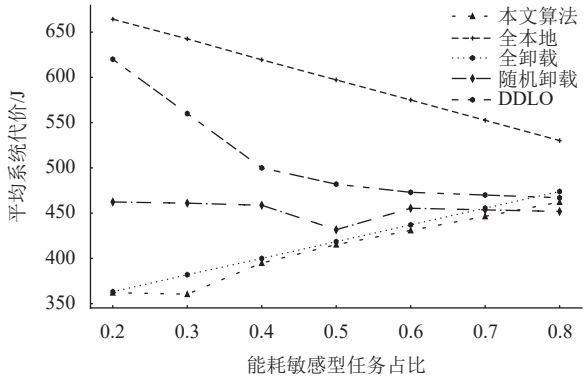


图 8 不同类型任务占比下的系统代价

综上所述，本文算法在不同 VEC 环境下，均能够有效降低整体的系统代价，对比其余 4 种算法，本文算法能够将系统性能提升 0.53%～83.4%。对于全本地执行算法，较低的车辆 CPU 计算能力将会带来巨大的时延损耗，从而导致任务处理时延超过最大容忍时延最终导致任务计算失败，因此该算法在不同环境下的表现均较差。全卸载算法对比其余算法，利用 MEC 以及云服务器的高计算资源很大程度上降低了时延，但服务器一旦出现资源分配不足或者处理任务过多的情况，将会使车辆用户产生额外的等候时延与传输时延从而增大代价。对于随机卸载算法，卸载决策一直没有办法达到最优决策，因此整体的效果也不够理想。DDLO 算法不考虑任务类型划分，因此卸载决策不具备针对性，整体算法效果略差。本文算法中的 DNN 完成多次学习以后会逐渐收敛，能够在极短时间内使卸载决策接近最优解，有效降低了 VEC 系统的能耗与时延。

4 结束语

本文提出了一种基于深度学习的并行式 VEC 网络卸载方案，该方案可以有效降低 VEC 系统的能耗与车联网用户的平均服务时延。本文首先考虑根据车载任务本身的需求、车辆剩余电量、路段任务数量等因素将车载任务进行判别划分；其次基于云−边−端三者的协同工作建立多约束条件的 VEC 系统代价模型；然后利用卸载算法中的调度单元以及罚函数消除约束条件；最后，利用多个卸载单元中的 DNN 不断学习训练生成最优卸载决策，完成问题的求解。仿真实验结果表明，本文提出的卸载算法在不同 VEC 环境下具有比较稳定的表现，且有效降低了系统的能耗与时延。该算法中的神经网络经过训练，可以在不到一秒时间内生成最优卸载决策。之后，将在此单 MEC 服务器的基础上，研究如何使得不同路段的 MEC 服务器相互有效协作，缓解个别服务器计算资源有限的窘状。

参 考 文 献

[1] SHI W, CAO J, ZHANG Q, et al. Edge computing: Vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

[2] ZHANG K, MAO Y, LENG S, et al. Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading[J]. IEEE Vehicular Technology Magazine, 2017, 12(2): 36-44.

[3] CISCO V. Cisco visual networking index: Forecast and trends[EB/OL]. [2022-08-23]. https://www.cisco.com/c/zh_tw/about/news-center/news-20190225.html.

[4] ZHANG J, ZHAO X. An overview of user-oriented computation offloading in mobile edge computing[C]// 2020 IEEE World Congress on Services (SERVICES). [S.l.]: IEEE, 2020: 75-76.

[5] STORCK C R, DUARTE-FIGUEIREDO F. A survey of 5G technology evolution, standards, and infrastructure associated with vehicle-to-everything communications by internet of vehicles[J]. IEEE Access, 2020, 8: 117593-117614.

[6] KOMBATE D. The Internet of vehicles based on 5G communications[C]//2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). [S.l.]: IEEE, 2016: 445-448.

[7] KUTILA M, PYYKONEN P, HUANG Q, et al. C-V2X supported automated driving[C]//2019 IEEE International Conference on Communications Workshops (ICC Workshops). [S.l.]: IEEE, 2019: 1-5.

[8] PU L, CHEN X, MAO G, et al. Chimera: An energy-efficient and deadline-aware hybrid edge computing framework for vehicular crowdsensing applications[J]. IEEE Internet of Things Journal, 2018, 6(1): 84-99.

[9] LI S, GE H, CHEN X, et al. Computation offloading strategy for improved particle swarm optimization in mobile edge computing[C]//2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA). [S.l.]: IEEE, 2021:375-381.

[10] FENG W Y, YANG S Z, GAO Y, et al. Reverse offloading for latency minimization in vehicular edge computing[C]//ICC 2021 - IEEE International Conference on Communications. [S.l.]: IEEE, 2021: 1-6.

[11] REN Y L, CHEN X Y, GUO S, et al. Blockchain-based VEC network trust management: A DRL algorithm for vehicular service offloading and migration[J]. IEEE Transactions on Vehicular Technology, 2021, 70(8): 8148-8160.

[12] LIU Z, ZHAO J. Optimized task offloading policy in given sequence in mobile edge computing[C]//2020 IEEE 6th International Conference on Computer and Communications (ICCC). [S.l.]: IEEE, 2020: 1656-1660.

[13] ZENG F, CHEN Q, MENG L, et al. Volunteer assisted collaborative offloading and resource allocation in vehicular edge computing[J]. IEEE Transactions on Intelligent Transportation Systems, 2020, 22(6): 3247-3257.

[14] WU H, WOLTER K, JIAO P, et al. EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing[J]. IEEE Internet of Things Journal, 2020, 8(4): 2163-2176.

[15] MA C, ZHU J, LIU M, et al. Parking edge computing: Parked-Vehicle-Assisted task offloading for urban VANETs[J]. IEEE Internet of Things Journal, 2021, 8(11): 9344-9358.

[16] GAO M, SHEN R, SHI L, et al. Task partitioning and offloading in DNN-task enabled mobile edge computing networks[J]. IEEE Transactions on Mobile Computing, 2023, 22(4): 2435-2445.

[17] ZHAN W, LUO C, WANG J, et al. Deep-Reinforcement-Learning-Based offloading scheduling for vehicular edge computing[J]. IEEE Internet of Things Journal, 2020, 7(6): 5449-5465.

[18] LIU H, ZHAO H, GENG L, et al. A distributed dependency-aware offloading scheme for vehicular edge computing based on policy gradient[C]//2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom). [S.l.]: IEEE, 2021: 176-181.

[19] 方晨, 郭渊博, 王一丰, 等. 基于区块链和联邦学习的边缘计算隐私保护方法[J]. 通信学报, 2021, 42(11): 28-40.
FANG C, GUO Y B, WANG Y F, et al. Edge computing privacy protection method based on blockchain and federated learning[J]. Journal on Communications, 2021, 42(11): 28-40.

[20] ISLAM S, BADSHA S, SENGUPTA S. Context-aware fine-grained task scheduling at vehicular edges: An extreme reinforcement learning based dynamic approach[C]//2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM). [S.l.]: IEEE, 2021: 31-40.

[21] PARK S, KANG Y, TIAN Y, et al. Fast and reliable offloading via deep reinforcement learning for mobile edge video computing[C]//2020 International Conference on Information Networking (ICOIN). [S.l.]: IEEE, 2020: 10-12.

[22] HUANG L, FENG X, FENG A, et al. Distributed deep learnging-based offloading for mobile edge computing networks[J]. Mobile Networks and Applications, 2022, 27(3): 1123-1130.

编　辑　税　红