论文 Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems 复现

摘要

在移动边缘计算系统中，当大量移动设备将其任务卸载到边缘节点时，边缘节点可能会具有高负载。这些卸载的任务可能会遇到较大的处理延迟，甚至在截止日期到期时被丢弃。由于边缘节点的负载动态不确定，每个设备都很难以去中心化的方式确定其卸载决策（即是否卸载，以及应该将任务卸载到哪个边缘节点）。在这项工作中，文章考虑了不可整除和延迟敏感的任务以及边缘负载动态，并制定了一个任务卸载问题，以最小化预期的长期成本。文章提出了一种基于无模型的深度强化学习分布式算法，其中每个设备都可以在不知道其他设备的任务模型和卸载决策的情况下确定其卸载决策。为了改进算法中对长期成本的估计，文章采用了 long short-term memory (LSTM)，dueling deep Q-network (DQN) 和 double-DQN 技术。

关键词：移动边缘计算；计算卸载；资源分配；深度强化学习

1 引言

如今，移动设备负责处理许多计算密集型任务，例如数据处理和人工智能。尽管移动设备不断发展，但由于计算资源有限，这些设备可能无法以低延迟在本地处理所有任务。为了促进高效的任务处理，引入了移动边缘计算（MEC）[1]，也称为雾计算 [2] 和多访问边缘计算 [3]。MEC 有助于移动设备将其计算任务卸载到附近的边缘节点进行处理，以减少任务处理延迟。它还可以减少那些对延迟敏感的任务的丢弃任务的比例。在上述 MEC 系统下，由于任务之间复杂的交互，很难应用博弈论和在线优化等传统方法。为了应对这些挑战，在这项工作中，本文建议使用深度 Q 学习 [4]，这是一种无模型的深度强化学习（DRL）技术。这种方法使代理能够根据局部观察做出决策，而无需了解系统建模和动态。我们的目标是提出一种基于 DRL 的分布式算法，解决边缘节点的未知负载动态。它使每个移动设备能够在不知道其他移动设备的信息（例如任务模型、卸载决策）的情况下做出其卸载决策。在这项工作中，我们考虑了边缘节点的未知负载水平动态，并为 MEC 系统提出了一种基于 DRL 的分布式卸载算法。在所提出的算法中，每个移动设备可以利用本地观察到的信息以分散的方式确定卸载决策，包括其任务的大小、其队列的信息以及边缘节点的历史负载水平。

2 相关工作

在 MEC 中，有两个与任务卸载相关的主要问题。第一个问题是移动设备是否应该将其任务卸载到边缘节点。第二个问题是，如果移动设备决定执行卸载，那么设备应该将其任务卸载到哪个边缘节点。为了解决这些问题，一些现有的工作提出了任务卸载算法。有研究者在 [5] 中提出了一种算法来确定卸载决策以最大化网络收入。而 [6] 考虑了无线供电的 MEC 场景，并提出了一种优化卸载和电力传输决策的算法。在工作 [5]、[6] 中，每个设备从边缘节点获得的处理能力与卸载到边缘节点的任务数量无关。然而，实际上，边缘节点的处理能力可能有限，因此边缘节点分配给移动设备的处理能力取决于边缘节点处的负载水平（即卸载到边缘节点的并发任务的数量）。当大量移动设备将其任务卸载到同一边缘节点时，该边缘节点的负载可能很高，因此这些卸载的任务可能会经历较大的处理延迟。有些任务甚至可能在截止日期到期时被放弃。一些现有的工作已经解决了边缘节点的负载水平并提出了集中式任务卸载算法。另外有学者在 [7] 中提出了一种算法，考虑到移动设备不确定的计算要求，优化移动设备的卸载决策。而在 [8] 中，重点关注延迟敏感任务，并提出了一种在任务期限约束下最小化任务卸载能耗的算法。在 [9] 中为软件定义的超密集网络设计了一种集中式算法，以最大限度地减少任务延迟。在 [10] 中研究了任务卸载和路由的联合优化。然而，[7]、[8]、[9]、[10] 中这些集中式算法的运行可能需要系统的完整信息。

其他工作提出了考虑边缘节点的负载水平的分布式任务卸载算法，其中每个移动设备以分散的方式做出卸载决策。设计这样的分布式算法具有挑战性。这是因为当设备做出卸载决策时，该设备事先并不知道边缘节点处的负载水平，因为负载还取决于其他移动设备的卸载决策。此外，边缘节点的负载水平可能随时间而变化。为了应对这些挑战，有学者在 [11] 中，重点关注可分任务，提出了一种基于 Lyapunov 的算法来确保任务队列的稳定性。在 [12] 中考虑了移动设备间的策略性卸载交互，提出了一种基于价格的分布式算法。在 [13] 中设计了一种潜在的基于游戏的卸载算法，以最大限度地提高每个设备的体验质量。在 [14] 中设计了一种基于 Stackelberg 游戏的分布式算法。在 [15] 中提出了一种分布式卸载算法来解决移动设备之间的无线信道竞争。在 [16] 中提出了一种基于估计的方法，其中每个设备根据估计的处理和传输容量做出卸载决策。在 [17] 中提出了一种基于在线优化技术的算法来最小化任务的最大延迟。

在这项工作中，本文关注 MEC 系统中的任务卸载问题，并提出一种分布式算法来解决边缘节点的未知负载水平动态。与上述工作 [11]、[12]、[13]、[14]、[15]、[16]、[17] 相比，作者考虑了一个不同的、更现实的 MEC 场景。首先，现有的工作 [11] 考虑了可分割的任务（即任务可以任意分割），由于任务中各个位之间的依赖性，这可能不现实。尽管作品 [12]、[13]、[14]、[15]、[16] 考虑了不可分割的任务，但他们没有考虑底层的排队系统。因此，每个任务的处理和传输应该总是在一个时隙内（或在下一个任务到达之前）完成，但这在实践中并不总是能得到保证。与这些工作 [11]、[12]、[13]、[14]、[15]、[16] 不同，作者将不可分割的任务与排队系统一起考虑，并考虑处理和传输的实际场景一个任务可以持续多个时间段。这种情况处理起来很有挑战性，因为当新任务到达时，其延迟可能会受到前一个时隙中到达的其他设备的任务决策的影响。其次，与相关工作 [11]、[12]、[13]、[14]、[15]、[16]、[17] 考虑延迟容忍任务不同，研究关注具有处理期限的延迟敏感任务。这个问题很难解决，因为截止日期会影响边缘节点的负载水平，从而影响卸载任务的延迟。

3 本文方法

3.1 概述

研究针对移动边缘计算系统（MEC）中的关键问题——任务卸载，提出了一种基于深度强化学习（DRL）的智能决策框架。任务卸载的决策过程涉及到在资源受限的移动设备与性能强大的边缘服务器之间做出选择，以实现系统性能的最优。

本文复现工作一：深入理解环境模型并构建模拟 MEC 环境的模型，该模型综合了移动设备、边缘服务器以及它们之间的网络连接。在这一环境中，状态空间包含移动设备当前的任务大小、队列信息和边缘节点的负载水平历史等关键信息。动作空间则定义了移动设备可采取的行动，包括在本地执行任务、将任务卸载到边缘服务器，或者保持任务等待。奖励函数反映任务卸载决策对系统性能的影响，涵盖了能耗、延迟和系统吞吐量等多个维度。

本文复现工作二：深入理解任务块模型，包含移动设备模型和边缘节点模型。移动设备的计算任务都是不可分割的，可以选择本地处理或卸载到边缘节点处理。假设每个时隙开始时，移动设备有一定概率有新任务到达，当移动设备有新任务到达时，首先需要决定是本地处理任务还是将其卸载到边缘节点。如果决定本地处理，调度器将任务放入计算队列，如果决定卸载，调度器将任务放入传输队列，然后通过无线链路发送到选定的边缘节点。计算队列和传输队列都以先进先出（FIFO）的方式运作，假设如果任务在时隙内完成处理或传输，那么队列中的下一个任务将在下一个时隙开始时被处理或传输。每个边缘节点维护 1 个队列，每个队列对应集合中的一个移动设备，移动设备 m 的任务在时隙 t 被卸载在边缘节点时，该任务将在下一个时隙的开始被放入对应的队列。

本文复现工作三：学习并利用原文深度强化学习算法，用于学习在这一复杂环境中的最优任务卸载策略。通过与环境的交互，算法能够学习到在不同状态下采取何种动作以最大化长期累积奖励。本文将复现这一学习过程，并评估所学习策略的性能，通过模拟实验与现有方法进行比较，展示 DRL 方法在任务卸载决策中的优势。

3.2 模型

考虑了一个 MEC 系统中的边缘节点集合 $\mathcal{N} = \{1, 2, ..., N\}$ 和移动设备集合 $\mathcal{M} = \{1, 2, ..., M\}$，一个包含一系列时隙的集合 $\mathcal{T} = \{1, 2, ..., T\}$ ，每个时隙持续 $\triangle$ 秒；

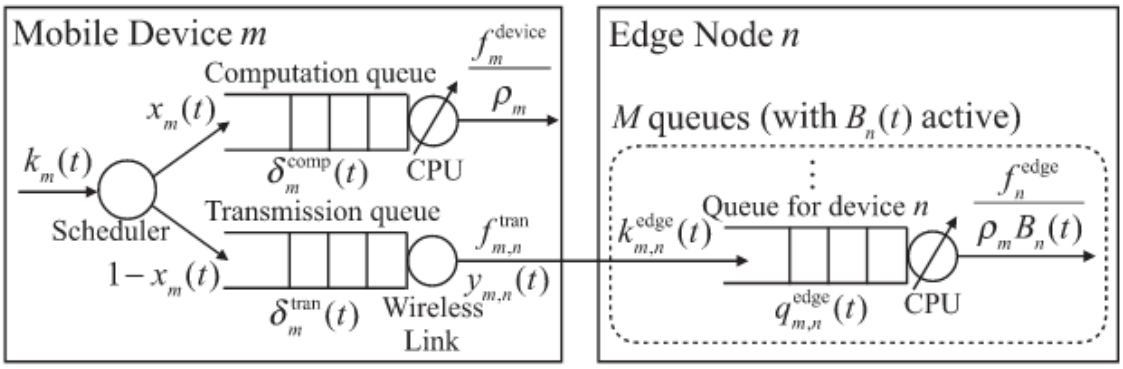

图 1. 具有移动设备 $m \in M$ 和边缘节点 $n \in N$ 的 MEC 系统。

3.2.1 移动设备模块

任务模型描述：用于模拟移动设备在不同时隙中新到达任务的处理情况；

（1）任务有否：文章定义了变量 $\|_{\Updownarrow}(\sqcup)$ ，表示在时隙 t 的开始，移动设备 m 有新任务到达；若没有新任务到达则 $\|_{\Updownarrow}(\sqcup)=0$；

（2）任务大小：再定义 $\lambda_m(t)$ 表示在时隙 t 开始时新到达任务的大小，如果存在新任务 $\|_{\Updownarrow}(\sqcup)$ 在时间槽 t 开始时到达，则 $\lambda_m(t)$ 等于任务 $k_m(t)$ 的大小；否则 $\lambda_m(t)$ 被设置为 0；

（3）任务处理密度：任务 $k_m(t)$ 需要一个处理密度 ρ_m （以每比特 CPU 周期数计），即处理单位数据所需要的 cpu 周期数；

（4）任务截至时间：任务 $k_m(t)$ 有一个截止时间 τ_m （以时隙计），如果任务 $k_m(t)$ 在时隙 $t+\tau_m-1$ 结束之前没有被完全处理，则他将被立即丢弃；

任务卸载决策：移动设备在时隙 $t\in\mathcal{T}$ 开始时对新到达的任务 $k_m(t)$ 进行任务卸载决策的过程，即是否在本地处理或卸载到边缘节点，以及如何决定卸载到哪个边缘节点；

（1）处理决策变量：文章定义二元变量 $x_m(t)\in\{0,1\}$ 来表示任务 $k_m(t)$ 是否在本地处理或卸载到边缘节点，如果任务在本地处理，则 $x_m(t)=1$ ；如果任务卸载到边缘节点，则 $x_m(t)=0$ ；

（2）数据到达队列：在时隙 t 开始时，用 $\lambda_m(t)x_m(t)$ 表示到达移动设备 m 计算队列的比特数 (大小)，$\lambda_m(t)(1-x_m(t))$ 表示到达移动设备 m 传输队列的比特数（大小）；

（3）卸载决策变量：如果任务 $k_m(t)$ 需要卸载到边缘节点，定义二元变量 $y_{m,n}(t)\in\{0,1\}$ 来表示任务是否卸载到边缘节点 $n\in\mathcal{N}$ 。如果任务卸载到边缘节点 n ，则 $y_{m,n}(t)=1$ ，否则 $y_{m,n}(t)=0$ ；

（4）卸载约束：每个任务只能卸载到一个边缘节点，即：

$$\sum_{n\in\mathcal{N}} y_{m,n}(t)=\mathbf{1}(x_m(t)=0),\quad m\in\mathcal{M},t\in\mathcal{T}, \tag{1}$$

计算队列：移动设备如何处理新到达的任务，包括任务在计算队列中的等待时间和处理时间，以及如何确定任务完成或被丢弃的时隙；

（1）机制：计算队列按照先进先出（FIFO）的方式运作，到达的任务是那些需要在本地处理的任务。定义 f_m^{device} (以 cpu 周期每秒计) 常数来表示移动设备 m 的 cpu 处理能力，且如果任务 $k_m(t)$ 被放入计算队列，定义 $l_m^{\text{comp}}(t)\in\mathcal{T}$ 表示任务 $k_m(t)$ 被处理或丢弃的时隙，如果任务 $k_m(t)$ 没有被放入计算队列或 $k_m(t)=0$ ，则 $l_m^{\text{comp}}(t)=0$ ；

（2）任务等待时间：让 $\delta_m^{\text{comp}}(t)$ (以时隙计) 表示任务 $k_m(t)$ 在计算队列中等待处理的时隙数，移动设备 m 在决定将任务放入哪个队列之前会计算 $\delta_m^{\text{comp}}(t)$ ，且给定 t' <t，$\delta_m^{\text{comp}}(t)$ 的计算公式为：

$$\delta_m^{\text{comp}}(t)=\left[\max_{t'\in\{0,1,\ldots,t-1\}} l_m^{\text{comp}}(t')-t+1\right]^+ \tag{2}$$

（3）任务完成时间：如果移动设备 m 在时隙 t 开始时将任务 $k_m(t)$ 放入计算队列（即 $x_m(t)=1$ ），则任务 $k_m(t)$ 将在时隙 $l_m^{\text{comp}}(t)$ 被处理或丢弃，其中 $\lceil\cdot\rceil$ 是向上取整：

$$l_m^{\text{comp}}(t)=\min\left\{t+\delta_m^{\text{comp}}(t)+\left\lceil\frac{\lambda_m(t)}{f_m^{\text{device}}\Delta/\rho_m}\right\rceil-1,\ t+\tau_m-1\right\}, \tag{3}$$

传输队列：移动设备如何处理需要卸载到边缘节点的任务，包括任务在传输队列中的等待时间和传输时间，以及如何确定任务完成传输或被丢弃的时隙；

（1）机制：传输队列按照先进先出（FIFO）的方式运作，到达的任务是需要卸载到边缘节点的任务。移动设备（表示为集合 $\mathcal{M}$ ）通过正交信道与边缘节点（表示为集合 $\mathcal{N}$ ）进行通信，且无线传输受到路径损耗和小规模衰落的影响；

（2）传输速率：从移动设备 m 到边缘节点 n 的传输速率，用 $r_{m,n}^{tran}$ 表示（以比特每秒为单位），计算如下：

$$r_{m,n}^{\text{tran}} = W \log_2 \left(1 + \frac{|h_{m,n}|^2 P}{\sigma^2}\right), \quad m \in \mathcal{N}, n \in \mathcal{N} \tag{4}$$

3.2.2 边缘节点模块

边缘节点的队列：每个边缘节点上的队列都按照先进先出（FIFO）的方式运作，队列的到达任务是移动设备卸载到该边缘节点的任务。定义 $q_{m,n}^{\text{edge}}(t)$ （以比特为单位）表示在时隙 t 结束时，移动设备 m 在边缘节点 n 上的队列长度；

（1）活跃队列：在时隙 t 中，移动设备 m 的任务到达队列（即 $\lambda_{m,n}^{\text{edge}}(t) > 0$ ）或者队列在时隙 t-1 结束时非空（即 $q_{m,n}^{\text{edge}}(t-1) > 0$ ），则移动设备 m 的队列为活跃队列，文章中定义 $\mathcal{B}_n(t)$ 来表示在时隙 t 中边缘节点 n 上的活跃队列集合：

$$\mathcal{B}_n(t) = \{m \mid \lambda_{m,n}^{\text{edge}}(t) > 0 \text{ or } q_{m,n}^{\text{edge}}(t-1) > 0,\ m \in \mathcal{M}\}. \tag{5}$$

又定义 $B_n(t)$ 来表示在时隙 t 中边缘节点 n 上活跃队列的数量，即 $B_n(t) = |\mathcal{B}_n(t)|$ ；

（2）任务处理模型和能力：文章中考虑移动设备的任务具有相同的优先级，每个边缘节点有一个 CPU 用于处理队列中的任务，采用广义处理器共享（Generalized Processor Sharing, GPS）模型，其中活跃队列平等共享 CPU 的处理能力，且由于活跃队列的数量 $B_n(t)$ 是随时间变化且事先未知的，分配给每个队列的处理能力可能会随时间变化；定义 f_n^{edge}（以 CPU 周期每秒计）表示边缘节点 n 的处理能力，让 $e_{m,n}^{\text{edge}}(t)$（以比特为单位）表示在时隙 t 结束时队列丢弃的任务比特数，且队列长度更新：

$$q_{m,n}^{\text{edge}}(t) = \left[q_{m,n}^{\text{edge}}(t-1) + \lambda_{m,n}^{\text{edge}}(t) - \frac{f_n^{\text{edge}}\Delta}{\rho_m B_n(t)} \mathbf{1}(m \in \mathcal{B}_n(t)) - e_{m,n}^{\text{edge}}(t)\right]^+. \tag{6}$$

任务处理或删除：移动设备在边缘节点的任务处理时间的确定和相关约束，如果移动设备 m 的任务 $k_{m,n}^{\text{edge}}(t)$ 在时间槽 t 被放入边缘节点 n 的对应队列，则定义变量 $l_{m,n}^{\text{edge}}(t) \in \mathcal{T}$ 表示任务被处理或丢弃的时隙，如果 $k_{m,n}^{edge}(t) = 0$ ，则 $l_{m,n}^{\text{edge}}(t) = 0$ ；

（1）开始处理时间：定义 $\hat{l}_{m,n}^{\text{edge}}(t)$ 表示任务 $k_{m,n}^{\text{edge}}(t)$ 开始处理的时隙，计算公式如下，其中 $l_{m,n}^{\text{edge}}(0) = 0$ ：

$$\hat{l}_{m,n}^{\text{edge}}(t) = \max\left\{t, \max_{t' \in \{0,1,\ldots,t-1\}} \left(l_{m,n}^{\text{edge}}(t') + 1\right)\right\}, \tag{7}$$

（2）处理时间约束：对于 $m \in \mathcal{M}$, $n \in \mathcal{N}$, $t \in \mathcal{T}$ ；任务 $k_{m,n}^{\text{edge}}(t)$ 的大小不能超过 $\hat{l}_{m,n}^{\text{edge}}(t)$ 到 $l_{m,n}^{\text{edge}}(t)$ 边缘节点 n 的分配给移动设备 m 的总处理能力：

$$\sum_{t'=l_{m,n}^{\text{edge}}(t)}^{l_{m,n}^{\text{edge}}(t)} \frac{f_n^{\text{edge}}\Delta}{\rho_m B_n(t')} \mathbf{1}(m \in \mathcal{B}_n(t')) \geq \lambda_{m,n}^{\text{edge}}(t), \tag{8}$$

而从 $\hat{l}_{m,n}^{\text{edge}}(t)$ 到 $\hat{l}_{m,n}^{\text{edge}}(t)-1$ 的总处理能力小于任务大小：

$$\sum_{t'=\hat{l}_{m,n}^{\text{edge}}(t)}^{l_{m,n}^{\text{edge}}(t)-1} \frac{f_n^{\text{edge}}\Delta}{\rho_m B_n(t')}\mathbf{1}(m\in\mathcal{B}_n(t')) < \lambda_{m,n}^{\text{edge}}(t). \tag{9}$$

3.3 算法

基于深度强化学习（DRL）的算法，为了减轻移动设备的计算负担，通过让边缘节点协助移动设备训练神经网络，让每个移动设备 m 都与一个特定的边缘节点 n_m 关联，该边缘节点帮助设备 m 进行训练。边缘节点的选择基于与移动设备的最大传输容量。算法在移动设备和边缘节点上分别执行，利用移动设备的经验（状态、动作、成本和下一个状态）来训练神经网络，从而获得状态-动作对到 Q 值的映射。

边缘节点为每个移动设备维护一个重放记忆 D_m，存储设备的经验。同时，边缘节点维护两个神经网络：评估网络 Net_m 用于动作选择，目标网络 $TargetNet_m$ 用于近似动作的预期长期成本，通过评估网络和目标网络，移动设备可以选择最小化预期长期成本的动作。目标网络的参数会定期更新，以保持与评估网络的同步，从而更好地近似长期成本。

3.3.1 在移动设备 m 上的 DRL 算法

Algorithm 1. DRL-Based Algorithm at Device $m \in \mathcal{M}$

```
for episode = 1, 2, ..., E do
    Initialize s_m(1);
    for time slot t ∈ T do
        if device m has a new task arrival k_m(t) then
            Send a parameter_request to edge node n_m;
            Receive network parameter vector θ_m;
            Select an action a_m(t) according to (22);
        end if
        Observe the next state s_m(t+1);
        Observe a set of costs {c_m(t'), t' ∈ T̃_{m,t}};
        for each task k_m(t') with t' ∈ T̃_{m,t} do
            Send (s_m(t'), a_m(t'), c_m(t'), s_m(t'+1)) to n_m;
        end for
    end for
end for
```

图 2. 基于设备 $m \in M$ 的 DRL 算法。

算法涉及多个 Episode，每个 Episode 开始时，移动设备 mm 都会初始化其状态，即：

$$s_m(1) = \left(\lambda_m(1), \delta_m^{\text{comp}}(1), \delta_m^{\text{tran}}(1), q_m^{\text{edge}}(0), H(1)\right), \tag{10}$$

当移动设备 m 有新任务到达时，它可以向关联的边缘节点 n_m 发送参数请求，以更新用于任务卸载决策的参数向量 $_m$。为了减少通信开销，这个请求不是在每个时隙都进行；移动设备 mm 根据评估网络 Net_m 的参数向量 $_m$ 选择动作，以最小化 Q 值。动作选择包括随机探索和基于 Q 值的贪婪选择：

$$a_m(t) = \begin{cases} \text{a random action from } \mathcal{A}, & \text{w.p. } \epsilon, \\ \arg\min_{a\in\mathcal{A}} Q_m(s_m(t), a; \theta_m), & \text{w.p. } 1-\epsilon, \end{cases} \tag{11}$$

在每个时隙结束时，即下一个时隙（t+1）开始时移动设备 m 观察下一个状态（$S_m(t-1)$）和与任务相关的成本，由于任务处理和传输可能持续多个时隙，取决于任务 $k_m(t)$ 延迟的成本 $C_m(t)$ 可能不会立即被观察到，而是在任务处理或传输完成后观察，但是此时设备可以观察到属于任务 $k_m(t')$ 的成本集合，定义为：

$$\widetilde{\mathcal{T}}_{m,t} = \left\{ t' \mid t' = 1, 2, \ldots, t, \lambda_m(t') > 0, l_m^{\text{comp}}(t') = t \text{ or } \sum_{n\in\mathcal{N}} \sum_{i=t'}^{t} \mathbf{1}(k_{m,n}^{\text{edge}}(i) = k_m(t')) l_{m,n}^{\text{edge}}(i) = t \right\}. \tag{12}$$

3.3.2 在边缘节点 n 上的 DRL 算法

Algorithm 2. DRL-Based Algorithm at Edge Node $n \in \mathcal{N}$

Initialize replay memory D_m for $m \in \mathcal{M}_n$ and Count := 0;
Initialize Net_m with random θ_m for $m \in \mathcal{M}_n$;
Initialize $Target\_Net_m$ with random θ_m^- for $m \in \mathcal{M}_n$;
while True **do**
  if receive a parameter_request from $m \in \mathcal{M}_n$ **then**
    Send θ_m to device m;
  end if
  if receive an experience $(\boldsymbol{s}_m(t), \boldsymbol{a}_m(t), c_m(t), \boldsymbol{s}_m(t+1))$ from $m \in \mathcal{M}_n$ and Converge_Indicator = 0 **then**
    Store $(\boldsymbol{s}_m(t), \boldsymbol{a}_m(t), c_m(t), \boldsymbol{s}_m(t+1))$ in D_m;
    Sample a set of experiences (denoted by $\mathcal{I}$) from D_m;
    for each experience $i \in \mathcal{I}$ **do**
      Obtain experience $(\boldsymbol{s}_m(i), \boldsymbol{a}_m(i), c_m(i), \boldsymbol{s}_m(i+1))$;
      Compute $\hat{Q}_{m,i}^{\text{Target}}$ according to (26);
    end for
    Set vector $\hat{\boldsymbol{Q}}_m^{\text{Target}} := (\hat{Q}_{m,i}^{\text{Target}}, i \in \mathcal{I})$;
    Update $\boldsymbol{\theta}_m$ to minimize $L(\boldsymbol{\theta}_m, \hat{\boldsymbol{Q}}_m^{\text{Target}})$ in (24);
    Count := Count + 1;
    if mod(Count, Replace_Threshold) = 0 **then**
      $\boldsymbol{\theta}_m^- := \boldsymbol{\theta}_m$;
    end if
  end if
end while

图 3. 基于设备 $n \in N$ 的 DRL 算法。

初始化、请求和响应：首先边缘节点 n 为移动设备 m 初始化重放记忆 D_m 和两个神经网络 Net_m 及 $TargetNet_m$，然后边缘节点 n 等待移动设备 m 的请求，接收到参数请求时，会发送 Net_m 的当前参数向量 $_m$ 给设备 m。同时，如果边缘节点 n 接收到移动设备 m 的经验数据 $(s_m(t), a_m(t), c_m(t), s_m(t+1))$，边缘节点会将其存储在 D_m 中。

并行、训练、损失函数：参数向量的回复和网络训练可以并行进行，无论训练是否进行中，边缘节点在接收到参数请求时会立即发送当前参数向量。边缘节点通过随机抽样重放记忆中的经验样本来训练神经网络，以最小化 Net_m 下的 Q 值和 $TargetNet_m$ 计算的目标 Q 值之间的差异来更新参数向量 $_m$，即通过边缘节点计算 $Q_m^{Target} = (\hat{Q}_{m,i}^{Target}, i\,\mathcal{I})$ 和最小化损失函数，损失函数如下：

$$L(\theta_m, \hat{Q}_m^{\text{Target}}) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \left(Q_m(s_m(i), a_m(i); \theta_m) - \hat{Q}_{m,i}^{\text{Target}} \right)^2. \tag{13}$$

最小化方法、双 DQN 技术：上式损失函数表征了在当前网络参数向量 $_m$ 下给定状态 $s_m(i)$ 下动作 $a_m(i)$ 的 Q 值和每个经验 $i\,\mathcal{I}$ 的目标 Q 值 $Q_{m,i}^{Target}$ 之间的差距，通过在神经网络上执行反向传播和迭代优化算法（如梯度下降算法）来完成损失函数的最小化；对于经验 $i\,\mathcal{I}$ 的目标 Q 值 $Q_{m,i}^{Target}$ 是基于双 DQN 技术确定的，此方法可以改善预期长期成本的估计；

目标 Q 值更新：针对目标 Q 值的推导，让 a_i^{Next} 表示在 Net_m 下给定状态 $s_m(i+1)$ 的最小 Q 值的动作，即：

$$a_i^{\text{Next}} = \arg\min_{a \in \mathcal{A}} Q_m(s_m(i+1), a; \theta_m). \tag{14}$$

且经验 i 的 $\hat{Q}_{m,i}^{Target}$ 值通过下式推导：

$$\hat{Q}_{m,i}^{\text{Target}} = c_m(i) + \gamma Q_m(s_m(i+1), a_i^{\text{Next}}; \theta_m^-). \tag{15}$$

目标 Q 值 $\hat{Q}_{m,i}^{Target}$ 揭示了给定状态 $s_m(i)$ 下动作 $a_m(i)$ 的预期长期成本，即经验 i 中的 cost 和在经验 i 中的下一个状态下可能被选择的动作的折扣 Q 值之和，该动作是在网络 $TargetNet_m$ 下进行；

4 复现细节

4.1 与已有开源代码对比

复现在源码基础上写了可视化文件，是用于绘制训练过程中的三个指标：平均成本、任务丢弃率和平均任务延迟。函数使用 matplotlib 库来绘制图形，并提供了保存图形和显示图形的选项。另外，提供一个脚本主要用于绘制移动边缘计算相关结果的图表。脚本提供了三种不同类型的绘图功能：成本图、丢弃率图和平均延迟图。另外基于此源码修改成功应用于自己的工作内容的模型中（最优数据包调度模型）。

4.2 实验环境搭建

软件环境：使用 Python 3.7 进行深度强化学习模型的实现，采用 TensorFlow 1.4.0 进行模型的训练和评估。

网络设置：构建一个包含多个边缘服务器和移动设备的网络拓扑结构，模拟真实世界的网络环境，设置不同的网络带宽、延迟和丢包率，以评估模型在不同网络条件下的性能。

5 实验结果分析

我考虑了一个具有 50 个移动设备和 5 个边缘节点的场景。参数设置见表 1。神经网络设置如下。批次大小设置为 16。学习率等于 0.001，折扣因子等于 0.9。随机探索的概率从 1 逐渐减小到 0.01。同时，我使用了 RMSprop 算法优化器。在这些模拟中，我关注的是一个固定环境的场景，即过渡函数（从状态和动作到下一个状态）和成本函数（从状态和动作到成本）不随时间变化。在非平稳环境下，如果环境发生了变化，那么原文提出的算法可以通过将随机探索的概率重置为 1 来适应环境，从而再次进行随机探索。

表 1. Parameter Settings

Parameter	**Value**
Δ	0.1 second
$\mathrm{f}_m^{\mathrm{device}}, m \in \mathcal{M}$	2.5 GHz [16]
$\mathrm{f}_n^{\mathrm{edge}}, n \in \mathcal{N}$	41.8 GHz [16]
$\mathrm{r}_{m,n}^{\mathrm{tran}}, m \in \mathcal{M}, n \in \mathcal{N}$	14 Mbps [18]
$\lambda_m(t), m \in \mathcal{M}, t \in \mathcal{T}$	{2.0, 2.1, ..., 5.0} Mbits [5]
$\rho_m, m \in \mathcal{M}$	0.297 gigacycles per Mbits [5]
$\tau_m, m \in \mathcal{M}$	10 time slots (i.e., 1 second)
Task arrival probability	0.3

5.1 算法收敛

所提算法中的神经网络是在线训练的，其中实时收集的经验用于训练神经网络并更新任务卸载决策。我在不同的神经网络超参数和算法设置下评估了所提算法的收敛性，考虑 1000 episodes，其中每 episode 有 100 个时隙。

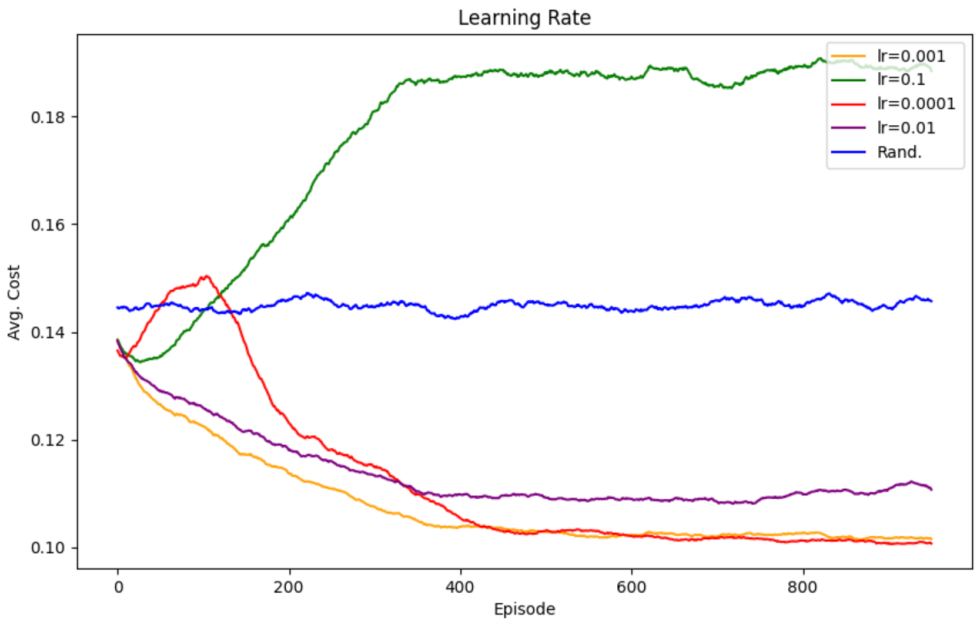

图 4. 算法在不同学习率下收敛性

图 4 显示了所提出的算法在不同学习率 (记为‘lr’) 值下的收敛性，其中学习率是每次迭代中向损失函数最小值移动的步长。在图 4 中，$lr = 10^{-3}$ 导致了相对较快的收敛速度和较小的收敛代价。当学习率 (即 10^{-4}) 较小时，收敛速度较慢。当学习率 (即 10^{-2}、10^{-1}) 较大时，收敛成本增加，甚至可能高于随机策略。

5.2 性能评估

性能指标：丢弃任务比率 (丢失任务的数量与总任务到达数量的比率) 和平均延迟 (已处理任务的平均延迟)。

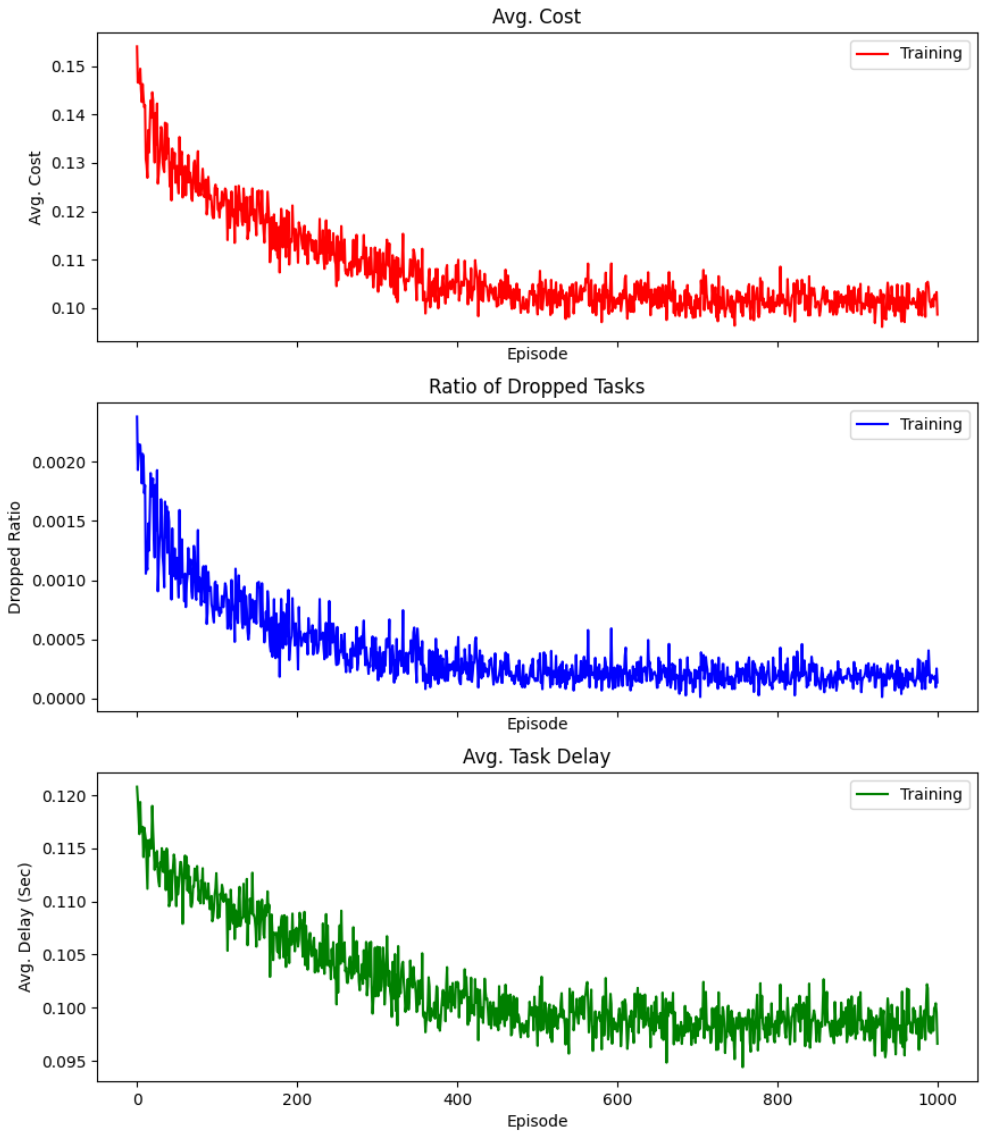

图 5. 算法在两个指标下的评估情况

6 总结与展望

未来有几个方向可以扩展这项工作。首先，扩展简化的无线网络模型以纳入传输错误和移动设备之间的干扰。其次，可以在演示系统中评估算法性能，在该系统下应解决许多实际问题（例如，实际计算任务）。第三，可以降低所提出算法的计算复杂度。这可以通过结合深度压缩 [19] 来减少神经网络中乘法运算的数量并结合迁移学习 [20] 来加速收敛。此外，当我们让边缘节点帮助移动设备训练神经网络时，当神经网络很大时，由于神经网络参数传输而导致的通信开销和可扩展性问题可能会成为一个问题。为了扩展这项工作，可以应用深度压缩 [19] 来减少神经网络的权重数量以及表示每个权重所需的位数。此外，为了增强所提出的算法在非平稳环境下的性能，结合非平稳环境的强化学习（例如 [21]）和终身强化学习 [22] 等技术是很有趣的。最后但并非最不重要的一点是，可以应用博弈论和多智能体强化学习技术来进一步理解移动设备之间的策略交互。这些技术可以合并到所提出的算法中，以进一步解决边缘节点的负载水平动态问题。

参考文献

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, (New York, NY, USA), p. 13–16, Association for Computing Machinery, 2012.

[3] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.

[6] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177–4190, 2018.

[7] N. Eshraghi and B. Liang, "Joint offloading decision and resource allocation with uncertain task computing requirement," in *IEEE INFOCOM 2019-IEEE conference on computer communications*, pp. 1414–1422, IEEE, 2019.

[8] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu, "Energy-efficient admission of delay-sensitive tasks for mobile edge computing," *IEEE Transactions on Communications*, vol. 66, no. 6, pp. 2603–2616, 2018.

[9] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[10] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 10–18, 2019.

[11] X. Lyu, W. Ni, H. Tian, R. P. Liu, X. Wang, G. B. Giannakis, and A. Paulraj, "Distributed online optimization of fog computing for selfish devices with out-of-date information," *IEEE Transactions on Wireless Communications*, vol. 17, no. 11, pp. 7704–7717, 2018.

[12] L. Li, T. Q. Quek, J. Ren, H. H. Yang, Z. Chen, and Y. Zhang, "An incentive-aware job offloading control framework for multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 63–75, 2019.

[13] H. Shah-Mansouri and V. W. Wong, "Hierarchical fog-cloud computing for iot systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, 2018.

[14] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2467–2475, IEEE, 2019.

[15] L. Yang, H. Zhang, X. Li, H. Ji, and V. C. Leung, "A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing," *IEEE/ACM transactions on networking*, vol. 26, no. 6, pp. 2762–2773, 2018.

[16] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "Uloof: A user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2660–2674, 2018.

[17] G. Lee, W. Saad, and M. Bennis, "An online optimization framework for distributed fog network formation with minimal latency," *IEEE Transactions on Wireless Communications*, vol. 18, no. 4, pp. 2244–2258, 2019.

[18] S. by Ookla, "Speedtest market reports: Canada average mobile upload speed based on q2-q3 2019 data," August 2024. [Accessed: Aug. 5, 2024].

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[20] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[21] E. Lecarpentier and E. Rachelson, "Non-stationary markov decision processes, a worst-case approach using model-based reinforcement learning," *Advances in neural information processing systems*, vol. 32, 2019.

[22] D. Abel, Y. Jinnai, S. Y. Guo, G. Konidaris, and M. Littman, "Policy and value transfer in lifelong reinforcement learning," in *International Conference on Machine Learning*, pp. 20–29, PMLR, 2018.