深圳大学考试答题纸

(以论文、报告等形式考核专用)

二〇 二四 ～二〇 二五 学年度第 一 学期

课程编号	1501990037	课程名称	智能网络与计算	主讲教师	车越岭	评分	
学　　号	2022150130	姓名	林宪亮	专业年级	大二		

教师评语：

题目：

基于 A3C 学习和残差循环神经网络的随机边缘云计算环境动态调度

《智能网络与计算》期末大报告

2024-2025 学年第一学期

论文题目：基于 A3C 学习和残差循环神经网络的随机边缘云计算环境动态调度

英文题目：Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks

姓　　名：林宪亮

学　　号：2022150130

演讲时间：2024.12.11

撰写报告时间：2024.12.12

填写说明：

1. 严格按照模板提供的一级大标题撰写报告，根据需要可自行添加二级、三级小标题。不要更改报告格式、字体等，确保报告格式的统一性。
2. 每个人独自完成、提交一份论文阅读报告。**同一小组提交的报告不可雷同！**
3. 大报告需要就论文中的关键技术、创新点、主要设计与优化、论文不足及潜在改进方向或研究方向，以及本人的对该研究论文思考等方面进行阐述。
4. **报告需同时提交电子版和纸质版。电子版提交时请转化成 PDF 格式**，并以“**期末大报告-姓名.pdf**”命名，在 BB 上提交。**纸质版交到计算机大楼 715 助教李健。**
5. 模板中灰色斜体字为说明，最终提交版本中需要删除。
6. 电子版报告提交截止时间为 2024 年 12 月 30 日中午 12 时，逾期提交的零分。纸质版请于 2024 年 12 月 31 日前提交。请尽快准备并提交。

摘　要

在物联网与云计算蓬勃发展背景下，边缘云协同计算至关重要。然而，其面临动态工作负载与资源异构难题，现有调度方法多有局限。本文聚焦于此，提出创新方案。利用 A3C 学习与残差循环神经网络（R2N2）构建动态调度模型。创新点在于：A3C 使模型快速适应环境变化，R2N2 有效捕捉任务时间特性及学习网络权重，提升决策精度与效率。经 iFogSim 和 CloudSim 模拟实验，在真实 Bitbrain 数据集验证，新模型性能卓越。能耗降低 14.4%，因优化资源分配减少闲置与过载能耗损失；响应时间缩短 7.74%，依任务特性精准调度资源，减少等待与处理耗时；SLA 违反率减少 31.9%，合理安排任务保障服务质量；成本降低 4.64%，提升资源利用率降低开支。调度开销仅 0.002%，可忽略不计，模型为边缘云任务调度提供高效可靠策略，推动其在复杂动态环境广泛应用与发展。

1. 研究背景及意义

1.1 研究背景与动机

物联网技术迅猛发展，海量设备接入网络产生庞大实时数据量。边缘云计算架构作为新兴数据处理范式兴起，其融合边缘计算与云计算优势，于数据源头附近处理部分任务，缓减云端压力、降传输延迟、保数据隐私与安全，提升系统整体响应性能。然而，边缘云环境复杂多变，工作负载随机波动显著。用户行为模式差异、设备故障频发、网络状况动态起伏等因素致任务量、资源需求随机变化。资源异构性突出，边缘节点与云服务器在计算、存储、网络能力及架构、性能指标上差异大，使资源分配调度困难重重。再者，边缘设备响应时间受自身性能、网络距离、带宽等因素制约，差异明显，传统调度方法未充分考量这些特性，难以有效应对复杂动态边缘云环境需求，在此背景下，创新调度模型与方法需求迫切。

1.2 研究意义

理论层面，本研究创新融合 A3C 学习与残差循环神经网络（R2N2）构建调度模型，丰富边缘云任务调度理论体系。拓展强化学习与深度学习融合应用范畴，为智能调度算法设计提供新思路，深化对复杂环境下任务资源匹配关系及动态调度策略优化机制理解。实践意义深远，新模型经模拟与真实数据集验证，有效降低能耗、缩短响应时间、减少 SLA 违反率、削减成本，提升边缘云系统运营效率与服务质量，增强企业市场竞争力。同时，模型普适性强，适用于多领域边缘云场景，为智慧交通、工业物联网、智能医疗等行业提供高效调度方案，有力推动物联网产业蓬勃发展，创造更大经济社会效益。

2. 国内外研究现状

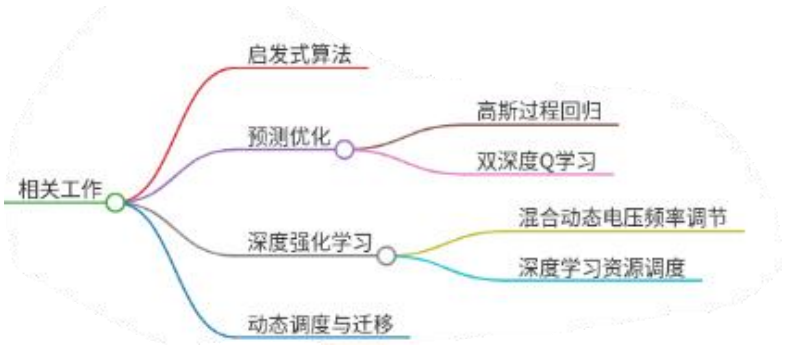


图 1 国内外研究现状

根据论文内容，CSDN，金锄头文库，51CTO 博客，腾讯云，电子发烧友等网站搜集的信息。

边缘云计算近年来受到广泛关注。相关研究主要集中在以下方面：

（1）边缘计算环境下的任务调度：提出了多种调度策略，如基于改进粒子群、蚁群算法等。但这些传统算法在面对复杂的边缘云环境时存在一些不足，例如难以处理随机环境下的动态工作负载、多层资源异构性以及设备响应时间差异等问题。

（2）深度强化学习的应用：深度强化学习在云环境下的任务调度中有一定研究，如 A3C 算法等。然而，其在边缘云环境中的应用还需要进一步探索和优化。

（3）残差网络的使用：残差网络在深度学习中能解决更深网络的训练问题，但在边缘云任务调度中的应用相对较少。

（4）边缘计算的相关概念与技术：包括边缘计算的体系架构、范例、优势等方面的探讨。例如边缘计算将计算和数据存储移动到网络边缘，提高了数据传输性能和实时性；边缘计算与协同边缘计算存在差异等。

（5）云边界整合算法：旨在解决云计算和边缘计算之间的数据传输和处理效率问题，但在数据安全、算法效率和性能优化等方面仍面临挑战。

（5）轻量级神经网络模型：针对边缘设备计算资源有限的情况，研究了网络剪枝、量化、结构设计等方法来减少模型参数量和计算量，但存在缺乏理论指导、通用性和泛化能力不足以及离线设计难以适应动态环境等问题。

与本工作的不同及已有工作的不足：

（1）已有工作大多针对特定问题或场景进行研究，缺乏对随机边缘云环境的全面考虑。本工作则创新地结合 A3C 学习与残差循环神经网络，构建了动态调度模型，更能适应随机环境下的复杂情况。

（2）传统调度算法和部分深度强化学习方法在处理边缘云的随机工作负载、资源异构性和设备响应时间差异等方面的能力有限，而本工作的模型能有效降低能耗、缩短响应时间等，提升边缘云系统运营效率。

（3）现有轻量级神经网络模型设计方法存在一些问题，而本工作利用残差循环神经网络，在一定程度上可缓解这些问题，提高模型的性能和效果。

3. 研究内容

3.1 主要挑战及创新点

挑战：

（1）工作负载随机性：任务到达时间、任务量及任务类型随机波动，难以精准预测，传统调度方法难以动态适配。

（2）资源异构难题：边缘节点与云服务器在多维度资源特性上存在巨大差异，资源管理与任务调度需综合考量多种因素，协调难度大。

（3）响应时间差异：边缘设备性能参差不齐、网络状况多变，致使响应时间差异显著，如何平衡优化成为关键。

创新点：

（1）设计适用于边缘云环境的数据驱动深度强化学习调度架构系统模型。此模型为边缘云复杂任务调度构建基础框架，整合数据驱动与深度强化学习优势，精准感知处理任务与资源动态变化，优化调度策略提升系统性能。

（2）勾勒分散式环境通用异步学习模型。应对边缘云去中心化特性，异步学习机制允许多个智能体并行探索学习，加速收敛，使模型迅速适应环境变化，提升调度决策实时性与灵活性。

（3）提出基于策略梯度强化学习方法（A3C）的随机动态调度策略。A3C 高效处理任务随机到达、资源动态波动难题，依策略梯度优化调度策略，平衡任务执行效率与资源利用成本，增强系统鲁棒性与适应性。

（4）展示基于残差循环神经网络（R2N2）框架挖掘混合边缘云设置调度时间模式。R2N2 精准捕捉任务时间依赖与资源变化趋势，为调度决策供关键信息，提升任务分配准确性与系统响应性能。

（5）通过大量模拟实验证实所提方案优越性并对比多基线策略。实验验证方案于能耗、响应时间、SLA 违反率及成本多指标优势，对比凸显创新性与有效性，为实际应用奠定坚实基础。

3.2 技术路线及实现

算法的总体流程如下：

（1）用户，物联网设备把需求传输给资源管理器，资源管理器结合需求以及目前主机状态生成三份特征表示传输给深度强化模型。

（2）系统中有多个智能体，每个智能体都有一份模型参数副本，模型根据特性进行决策，并计算上一个决策步本地的损失函数积累梯度，累计 12 个时间步就对全局的参数进行一次更新。

（3）资源管理器结合模型输出决策以及约束条件输出最终的决策结果发送给边云设备进行计算。

（4）边云设备把计算结果传输给资源管理器以及用户端，资源管理器根据计算结果计算新的损失，用户得到反馈。

下面是详细的技术细节：

系统模型与问题描述

图 2 结构图 1

系统架构：

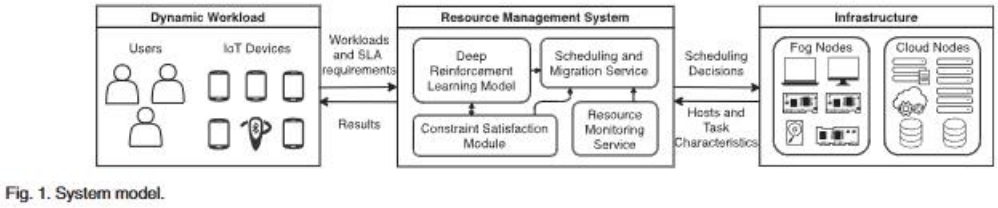


Fig. 1. System model.

图 3 系统架构图

系统建模的三个部分

（1）Dynamic Workload（动态工作负载）：

Users（用户）：代表使用系统的人员。

IoT Devices（物联网设备）：代表连接到系统的物联网设备。

Workloads and SLA requirements（工作负载和服务水平协议要求）：用户和物联网设备产生的工作负载以及相应的服务水平协议要求。

（2）Resource Management System（资源管理系统）：

Deep Reinforcement Learning Model（深度强化学习模型）：该模型用于做出调度和迁移决策。

Scheduling and Migration Service（调度和迁移服务）：根据深度学习模型的决策进行资源的调度和迁移。

Resource Monitoring Service（资源监控服务）：监控系统资源的使用情况。

Constraint Satisfaction Module（约束满足模块）：确保资源分配满足各种约束条件。

（3）Infrastructure（基础设施）：

Fog Nodes（雾节点）：靠近数据源的边缘计算节点，用于处理和存储数据。

Cloud Nodes（云节点）：云计算节点，提供更强大的计算和存储能力。

Hosts and Task Characteristics（主机和任务特性）：描述了基础设施中的主机和任务的特性。

各个部分的交互：

（1）动态工作负载部分生成工作负载和服务水平要求传输给资源管理系统。

（2）深度强化学习模型接受信息并进行调度分配决策并把决策结果传输给资源调度和迁移服务器，该服务器结合资源健康服务器和约束模块传输来的信息进行最终的决策然后把决策结果传输给基础设施进行计算。

（3）基础设施计算后把结果以及自己的状态信息传输回给资源管理系统，计算结果会被传输回动态工作负载。

动态任务加载：

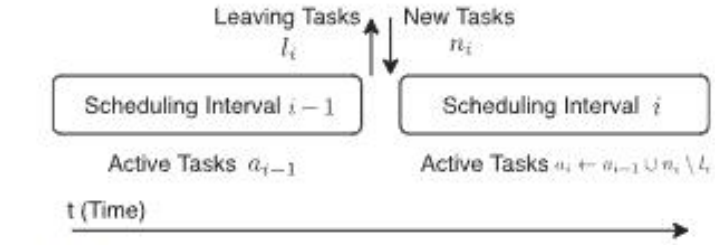


Fig. 2. Dynamic task workload model.

图 4 动态任务加载示意图

任务被划分为等时长的调度区间。调度区间依据出现顺序编号，如图 2 所示。第 i 个调度区间表示为 SI_i，起始于时间 i，持续至下一个区间起始时刻 t_{i+1}。在每个 SI_i 内，活跃任务是那些正在主机上执行的任务，记为 a_i。并且，在 SI_i 开始时，完成的任务集合记为 L_i，由工作负载生成模块（WGM）发送的新任务记为 n_i。任务 l_i 离开系统，新任务 n_i 加入系统。因此，在区间 SI_i 开始时，活跃任务 a_i 为 $a_i=a_i - 1 \cup n_i-l_i$。

问题转化：

可以把原始问题转化为下面这个优化问题：

$$\begin{aligned} \underset{Model}{\text{minimize}} \quad & \sum_i Loss_i \\ \text{subject to} \quad & \forall\, i,\ Action_i = Model(State_i) \\ & \forall\, i\, \forall\, T \in m_i \cup n_i, \{T\} \leftarrow Action_i(T). \end{aligned} \tag{1}$$

此公式描述了一个优化问题及其约束条件：

目标函数：

$$\underset{Model}{minimize} \sum_i Loss_i$$

旨在最小化所有 i 对应的损失值 $Loss_i$ 之和。这里的损失值可能代表模型预测结果与实际情况的偏差程度，通过最小化损失来优化模型性能，使模

型在处理任务调度等任务时能更精准地做出决策。

约束条件：

$$\forall i, Action_i = Model\,(State_i)$$

表示对于所有的 i，动作 $Action_i$ 由模型 Model 依据状态 $State_i$ 生成。即模型依据当前系统的状态（如节点资源使用情况、任务队列状态等）来确定采取何种调度动作（如任务分配到特定节点、分配特定资源量等）。

$$\forall i \forall T \in m_i \cup n_i, \{T\} \leftarrow Action_i(T)$$

意味着对于所有的 i 以及任务集合 $m_i \cup n_i$ 中的每个任务 T，任务 T 依据动作 $Action_i(T)$进行处理。例如任务依据模型确定的调度策略被分配到相应节点执行或获得特定资源分配，确保任务处理遵循模型决策。

Reinforcement Learning Model

图 5 结构图 2

输入：

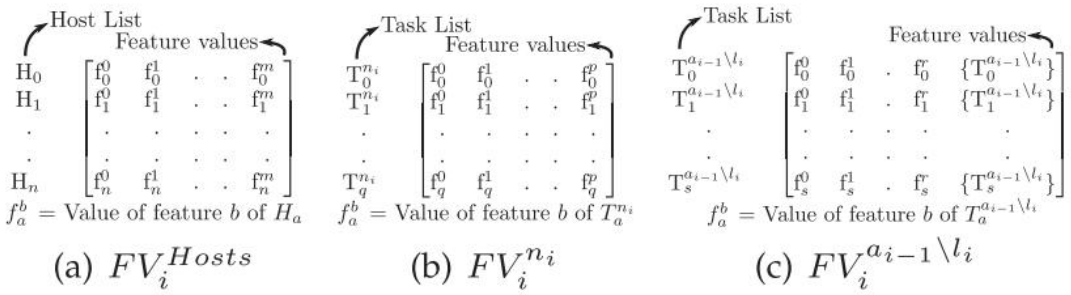


Fig. 4. Matrix representation of model inputs.

图 6 输入特征

输入由三部分组成：

（1）主机参数：这些参数包括 CPU（中央处理器）、内存（RAM）、带宽以及磁盘的利用率和容量。它还涵盖主机的功率特性、单位时间成本、每秒百万条指令数（MIPS）、响应时间以及分配到该主机的任务数量。不同的主机会有不同的计算能力（CPU）、内存容量（RAM）以及输入/输出（I/O）可用性（磁盘和带宽）。

（2）任务参数：任务被划分成两个互不相交的集合：n_i 和 a_{i-1} - l_i。前者包含诸如任务对 CPU、内存（RAM）、带宽以及磁盘的需求等参数。后者还包含在前一个时间区间中被分配的主机索引。这些任务集合的特征向量分别记为 FV_i^{ni} 和 $FV_i^{ai-1-li}$。

输出：

$$Action_i = \begin{cases} h \in Hosts \ \forall \ t \in n_i \\ h_{new} \in Hosts \ \forall \ t \in m_i \ if \ t \ is \ to \ be \ migrated \end{cases}$$

subject to

$Action_i$ is suitable for $t \ \forall \ t \in n_i \cup m_i$.

(2)

对于可迁移的任务，迁移到的新主机必须满足与这个任务适配。

$$\begin{aligned} Action_i(T_j^{a_i}) = H_j^k |\ T_j^{a_i} \in m_i \cup n_i \\ \wedge H_j^k \text{ is suitable for } T_j^{a_i} \\ \wedge \forall\ l\ <\ k, H_j^l \in Action_{i-1}^{PG}(T_j^{a_i}), \\ H_j^l \text{ is not suitable for } T_j^{a_i}. \end{aligned}$$

公式

$$Action_i(T_j^{a_i}) = H_j^k |\ T_j^{a_i} \in m_i \cup n_i$$

表明，对于属于可迁移任务集合与新任务集合中的任务，模型输出的动作将其分配到主机。这一分配建立在主机对任务适配性基础上，适配性涵盖计算资源（如满足任务计算复杂度需求）、存储资源（提供足够存储容量）及网络资源（保障数据传输要求）多维度考量。

后续条件

$$\wedge H_j^k \text{ is suitable for } T_j^{a_i}$$

进一步强化主机与任务适配要求，确保任务在分配主机上能高效执行，避免资源瓶颈阻碍任务进度。

$$\Lambda \forall l < k, H_j^l \in Action_{i-1}^{PG}\left(T_j^{a_i}\right), H_j^l \text{ is not suitable for } T_j^a$$

则体现决策的递进性与择优性。即对于任务，在先前决策中所涉及的主机均因资源适配不佳或其他限制因素无法满足任务需求，而新主机是经综合

评估筛选出的当前最优分配主机，此机制保障任务分配随系统动态变化持续优化，提升边缘云任务调度整体质量与效率。

损失函数：

损失函数由下面五个部分组成：

（1）平均能量消耗：

$$AEC_i^{Hosts} = \frac{\sum_{h \in Hosts} \alpha_h \int_{t=t_i}^{t_{i+1}} P_h(t)dt}{\sum_{h \in Hosts} \alpha_h P_h^{max}(t_{i+1} - t_i)}, \tag{5}$$

（2）平均响应时间：

$$ART_i = \frac{\sum_{t \in l_{i+1}} Response\ Time(t)}{|l_{i+1}| \max_i \max_{t \in l_i} Response\ Time(t)}. \tag{6}$$

（3）平均迁移时间：

$$AMT_i = \frac{\sum_{t \in a_i} Migration\ Time(t)}{|a_i| \max_i \max_{t \in l_i} Response\ Time(t)}. \tag{7}$$

（4）费用：

$$Cost_i = \frac{\sum_{h \in Hosts} \int_{t=t_i}^{t_{i+1}} C_h(t)dt}{\sum_{h \in Hosts} C_h^{max}(t_{i+1} - t_i)}, \tag{8}$$

（5）平均违规次数：

$$SLAV_i = \frac{\sum_{t \in l_{i+1}} SLA(t)}{|l_{i+1}|}. \tag{9}$$

对于上面五个部分的损失进行加权求和得到总的损失函数：

$$\begin{aligned} Loss_i = {} & \alpha \cdot AEC_{i-1} + \beta \cdot ART_{i-1} + \gamma \cdot AMT_{i-1} \\ & + \delta \cdot Cost_{i-1} + \epsilon \cdot SLAV_{i-1} \\ & \text{such that } \alpha, \beta, \gamma, \delta, \epsilon \geq 0 \\ & \wedge \alpha + \beta + \gamma + \delta + \epsilon = 1. \end{aligned} \tag{10}$$

惩罚项：

$$\begin{aligned} & Penalty_{i+1} = \\ & \frac{\sum_{t \in a_i} k | H^k = Action_i(t) \wedge H^k \in Action_i^{PG}(t)}{|a_i| \times n} \\ & + \frac{\sum_{t \in a_{i-1} \setminus l_i} \mathbb{1}(t \notin m_i \wedge Action_i(t) \neq \{t\})}{|a_i|}. \end{aligned} \tag{4}$$

第一个加数体现的是主机分配惩罚，第二个加数体现的是迁移惩罚。

总的损失函数：

$$Loss_i^{PG} = Loss_i + Penalty_i. \tag{11}$$

将损失函数加上惩罚性得到最终的损失函数。

模型更新流程：

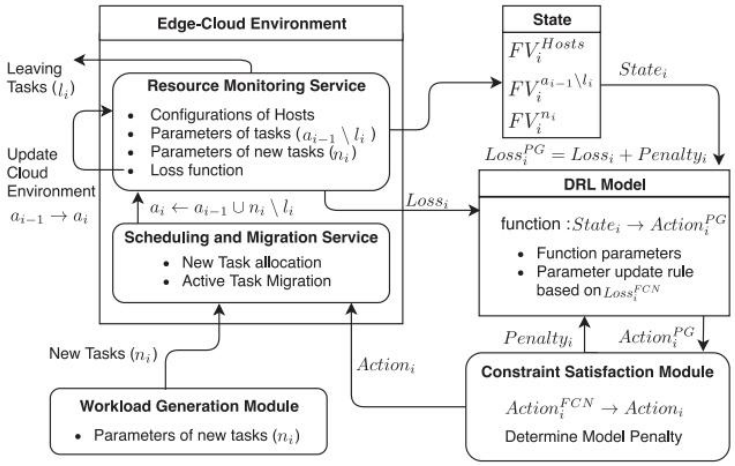


Fig. 6. Learning model.

图 7 模型总体架构图

在每个调度区间开始时，工作负载生成模块（WGM）会向调度与迁移服务（SMS）发送新任务。

然后，调度与迁移服务（SMS）和工作负载生成模块（WGM）会将 $State_i$ 发送给深度强化学习模型（DRLM），$State_i$ 包含主机的特征向量、上一区间剩余的活跃任务 $a_{i-1} - l_i$ 以及新任务 n_i 的相关信息。

同时，资源管理系统（RMS）会将损失值发送给深度强化学习模型（DRLM）。约束服务模块（CSM）会基于上一步决策发送惩罚值。 接着，模型会生成一个新的决策，并更新其参数，然后将其发送给约束服务模块（CSM）。

约束服务模块（CSM）会将模型的决策转换为可行的行动，并将其发送给资源管理系统（RMS）。它还会计算并存储下一个区间的惩罚值。

资源管理系统（RMS）会基于从约束服务模块（CSM）接收到的指令，分配新任务并迁移上一区间剩余的任务。

依次循环。

异步策略学习

图 8 结构图 3

循环神经网络结构：

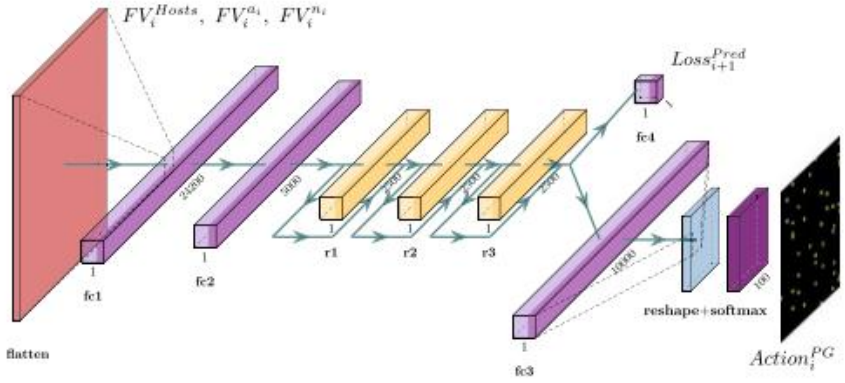


Fig. 7. Neural network architecture.

图 9 网络结构图

网络由展平层，全连接层，循环神经网络层，分类层组成。

$$e = \begin{cases} 0 \text{ if } max_{fc} = min_{fc} \\ min(1, max(0, \frac{e - min_{fc}}{max_{fc} - min_{fc}})) \text{ otherwise.} \end{cases} \tag{12}$$

输入的特征在输入到该神经网络前会进行归一化处理。

$$d\theta \leftarrow d\theta - \alpha \nabla_{\theta'} \log[\pi(State_i; \theta')](Loss_i^{PG} + CLoss_{i+1}^{Pred}) + \alpha \nabla_{\theta'}(Loss_i^{PG} + CLoss_{i+1}^{Pred} - CLoss_i^{Pred})^2. \tag{13}$$

该系统包含多个智能体，每个智能体都由该神经网络的一个参数副本，智能体会在本地计算积累梯度，然后在 12 个时间步数后根据上诉的公式进行全局参数的更新。θ 表示全局参数，θ'表示局部参数。

3.3 相关技术介绍

启发式算法

众多研究曾运用启发式或基于规则的策略调度边缘云环境任务。这类算法于一般状况下可发挥一定作用，但其缺陷显著。它未能考量工作负载及边缘云复合计算范式引发的动态变化，致使在边缘云环境中适应性欠佳。在实际应用中，面对网络状态波动、设备资源动态调整以及任务需求变更等复杂动态情境，启发式算法难以做出精准有效的调度决策，无法满足系统对资源高效利用与任务快速响应的需求。

强化学习技术

（一）基于值的强化学习方法深度 Q 学习（DQN）及其拓展的双深度 Q 学习（DDQN）等方法，借助神经网络近似 Q 值优化资源管理系统。在实践中，此类方法在高度随机的边缘云环境里效果不理想。它们常聚焦于单一 QoS 参数优化，像仅关注能耗或响应时间，却忽视其他关键指标。而且未运用异步更新机制提升在高随机环境的适应性，难以迅速响应环境变化。此外，这些方法未深入挖掘工作负载、网络及节点行为蕴含的时间模式，致使调度决策缺乏前瞻性与精准

性。同时，多采用集中式调度策略，在分散或分层的边缘云架构中，易引发单点故障、通信延迟及资源分配不均衡等问题，无法契合边缘云环境特性。

（二）策略梯度方法部分研究尝试的策略梯度方法存在局限性。在优化目标上，仅围绕单个 QoS 参数展开，未能全面综合考量能耗、响应时间、服务水平协议（SLA）违反率及成本等多元关键指标，致使调度方案顾此失彼。在应对边缘云复杂动态环境时，未充分发挥异步更新优势，无法及时精准调整策略，导致系统性能波动，难以保障任务高效稳定执行。

异步优势演员 - 评论家（A3C）学习 A3C

作为一种政策梯度法，通过异步运行多个演员代理脱颖而出。各代理配备独立神经网络并行训练，并周期性更新全局网络共享参数。此机制极大增强了对大状态 - 动作空间的探索能力，使其能迅速适应边缘云的随机环境变化。在边缘或云节点分散部署时，A3C 有效避免单点故障，确保系统可靠性与稳定性，为边缘云环境下分散式学习调度提供坚实技术支撑。

残差递归神经网络（R2N2）

R2N2 具备精准捕捉输入特征间复杂非线性关系及时间模式的卓越能力，其残差层大幅提升学习速率。与 A3C 协同构建的调度模型，依自适应损失函数与应用需求灵活优化 QoS 指标。通过高效策略学习最小化损失函数，R2N2 显著提升调度决策精准度与效率，充分利用时间模式信息优化任务分配与资源调度，有力提升系统整体性能。

4. 性能评估

图 10 结构图 4

数据集：BitBrain

该数据集拥有大量的代表真实世界基础设施使用模式的数据，这对模型精准的构建输入向量很有作用。

边缘节点和云节点设置：

边缘节点：intel i3 intel i5

云节点：intel Xeon

TABLE 2
Configuration of Hosts in the Experiment Set Up

Name	Processor	Core count	MIPS	RAM	Network Bandwidth	Disk Bandwidth	Cost Model	SPEC Power (Watts) for different CPU percentage usages										
								0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Edge Layer																		
Hitachi HA 8000	Intel i3 3.0 GHz	2	1800	8 GB	0.1 GB/s	76 MB/s	0.11 $/hr	24.3	30.4	33.7	36.6	39.6	42.2	45.6	51.8	55.7	60.8	63.2
DEPO Race X340H	Intel i5 3.2 GHz	4	2000	16 GB	1 GB/s	49 MB/s	0.23 $/hr	83.2	88.2	94.3	101	107	112	117	120	124	128	131
Cloud Layer																		
Dell PowerEdge R820	Intel Xeon 2.6 GHz	32	2000	48 GB	1 GB/s	49 MB/s	3.47 $/hr	110	149	167	188	218	237	268	307	358	414	446
Dell PowerEdge C6320	Intel Xeon 2.3 GHz	64	2660	64 GB	1.5 GB/s	1024 MB/s	6.94 $/hr	210	371	449	522	589	647	705	802	924	1071	1229

图 11 设备配置信息

评估指标：

（1）平均能量消耗
（2）平均响应时间
（3）平均违规次数
（4）总费用
（5）平均任务完成时间
（6）任务完成数量

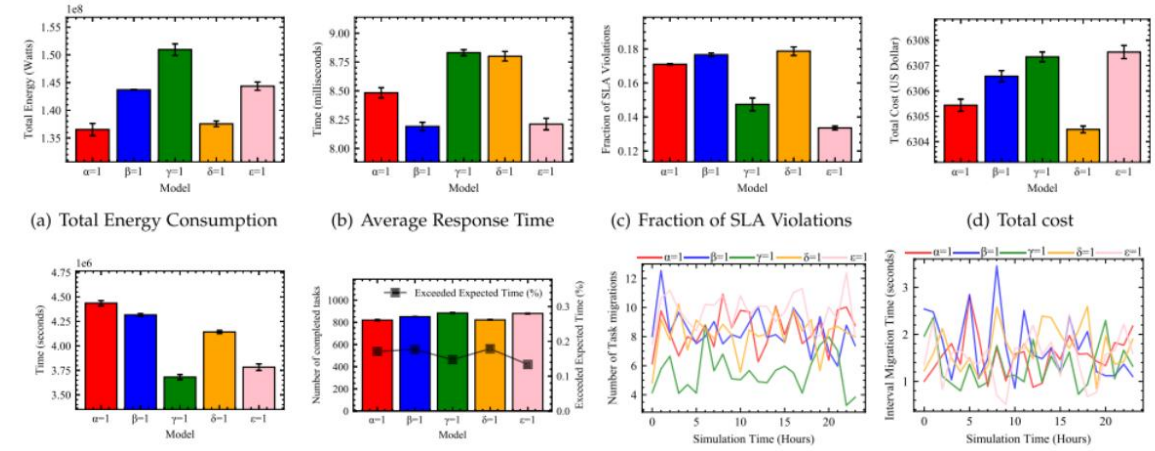
1) *Total Energy Consumption* which is given as $\sum_{h \in Hosts} \int_{t=t_i}^{t_{i+1}} P_h(t)dt$ for the complete simulation duration.
2) *Average Response Time* which is given as $\frac{\sum_{t \in l_{i+1}} Response\ Time(t)}{|l_{i+1}|}$.
3) *SLA Violations* which is given as $\frac{\sum_i SLAV_i \cdot |l_{i+1}|}{\sum_i l_i}$ where $SLAV_i$ is defined by Equation (9).
4) *Total Cost* which is given as $\sum_i \sum_{h \in Hosts} \int_{t=t_i}^{t_{i+1}} C_h(t)dt$.

基线模型：

LR-MMT：安排工作负载动态基于局部回归（LR）和最小迁移时间（MMT）启发式过载检测和任务选择
MAD-MC：调度动态的工作负载基于中值绝对偏差和最大相关策略启发式过载检测和任务选择
DDQN：标准的基于深度 q 学习的 RL 方法
DRL：基于策略梯度的强化方法与全连接的神经网络

实验结果：

调参实验：



(a) Total Energy Consumption (b) Average Response Time (c) Fraction of SLA Violations (d) Total cost

(e) Average Task Completion Time (f) Number of total completed tasks (g) Number of task migration in each interval (h) Total Migration Time in each interval

Fig. 9. Comparison of model trained with different loss functions.

图 12 参数实验结果

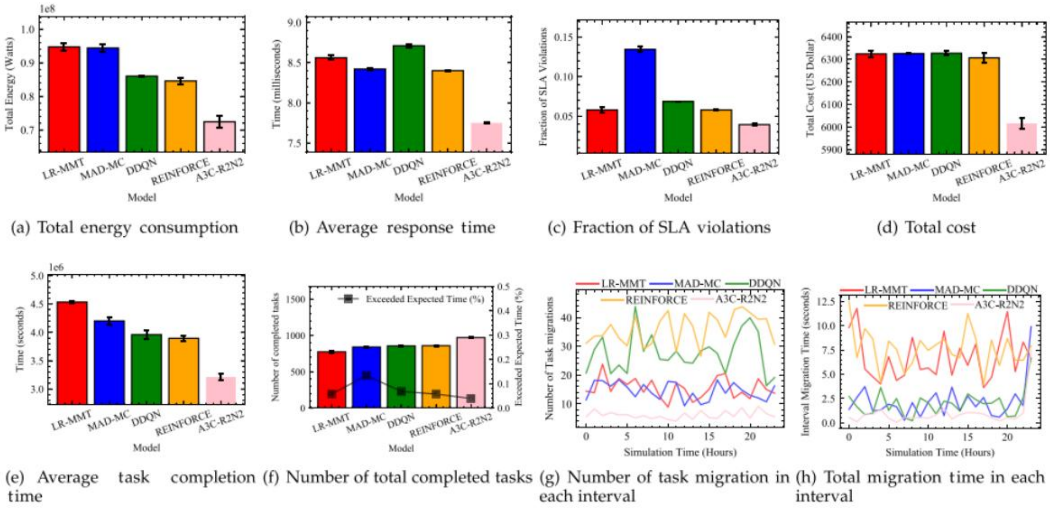
这个是对损失函数的权重参数的调参实验， β =1 表示除了 β 之外的权

重参数都设置为 0。从实验结果可以看出能量消耗与成本消耗成正相关，根据这些参数之间的关系可以帮助调参。

这是作者的调参结果：

$$(\alpha, \beta, \gamma, \delta, \epsilon) = (0.4, 0.16, 0.174, 0.135, 0.19). \quad (14)$$

对比实验：



(a) Total energy consumption (b) Average response time (c) Fraction of SLA violations (d) Total cost

(e) Average task completion time (f) Number of total completed tasks (g) Number of task migration in each interval (h) Total migration time in each interval

图 13 对比实验结果

这是作者提出的模型与其它基线模型的对比实验结果，从实验结果可以看出，作者提出的模型在六个评价指标下都优于其它的基线模型。

5. 未来工作及展望

作者创新性地结合了深度强化学习以及异步学习策略，在各个评价指标下都获得了进步，这是一个很有意义的工作，但我在阅读的过程中也发现了一些**不足之处**。

（1）训练成本高：正如作者所说，他们使用了 11 天进行训练和测试，那么当模型迁移到更复杂的环境，训练和测试的时间就会更长。并且强化学习调参是一件很麻烦的事情。

（2）深度学习模型结构单一：深度学习模型只使用了 RNN 和全连接，这在更复杂的特征提取任务上未必有很好的表现。

（3）实验环境局限性：模拟实验虽采用 iFogSim 和 CloudSim 及 Bitbrain 数据集，但与真实复杂物联网云环境仍存差异。现实中网络波动、设备故障突发、多类型任务复杂交互等未充分体现，使模型在实际应用效果存不确定性，如极端网络延迟下调度策略未验证，且模拟环境难以模拟所有现实场景，影响模型普适性。

（4）模型复杂度与可解释性权衡欠佳：A3C－R2N2 模型融合多种技术提升性能，但架构复杂致理解困难、可解释性弱。决策过程透明度低，企业运维人员难理解调度依据，不利模型优化推广。如多层神经网络权重意义不明，模型学习

与任务主机参数映射关系复杂模糊。

（5）安全隐私维度缺失：仅计划未来研究数据隐私安全，未深入探讨调度中数据加密、访问控制与任务调度协同，无法满足物联网敏感数据需求。如医疗物联网数据调度中，患者信息安全传输存储及访问权限管理未涉及，易致数据泄露风险。

改进建议：

（1）引入迁移学习和集成学习的技术，前期进行大规模数据下的大模型训练测试，后期仅仅对于个别网络层数进行微调，这样虽然前期训练成本高，但是后期实地应用时可以快速迁移。

（2）使用更复杂的模型结构，使用 CNN 来提取输入数据的时空特征，使用 transformer 架构捕捉时序信息，提高模型的能力。

（3）与物联网企业合作开展试点，于不同规模场景部署模型，收集多工况数据反馈优化。建立混合测试环境，融合模拟与真实数据，模拟极端网络条件，检验模型鲁棒性，提升应对复杂情况能力。

（4）引入可视化工具展示决策过程，如绘制任务调度路径图、节点资源分配热力图，直观呈现模型依据。结合领域知识简化模型或提取关键规则，增强运维人员对模型行为理解，便利故障诊断与策略调整，提高模型可信度。

（5）设计加密调度框架，确保任务数据全生命周期安全。构建访问控制模型，依任务主体属性权限分配资源访问权，结合区块链等技术记录验证数据访问操作，保障数据完整性、可追溯性，增强物联网云计算环境数据安全性。

6. 论文阅读心得

通过对该论文的阅读，以及对相关资料的查阅，提高了我对物联网领域，特别是边云计算的理解，了解了通过深度强化学习的知识原来也可以解决物联网方面的难题，这为我未来的科研方向又增加了一种选择。

通过全英的演讲，提供了我的英文演讲能力，这对我无疑是很有帮助的，如果以后有机会参加学术会议，我相信这是一次宝贵的英文演讲经验积累。

通过一门课让同学们了解一个研究的领域，激发同学们的科研兴趣，这是我对这门课意义的理解，这是很棒的一门课程，希望它越开越好。

7. 附录

附录包括了我总结的论文总体结构图，论文所提算法的复杂度分析，论文中算法的代码实现以及实验输出

论文总体结构图:

图 1 论文结构图

动态迁移算法的复杂度分析：

Algorithm 1. Dynamic Scheduling

Inputs:
1: Number of scheduling intervals N
2: Batch Size B
Begin
3: **for** interval index i from 1 to N **do**
4: **if** $i > 1$ and $i\%B == 0$ **then**
5: Use $Loss_i^{PG} = Loss_i + Penalty_i$ in RL Model for back-propagation
6: **end if**
7: send PREPROCESS($State_i$) to RL Model
8: $probabilityMap \leftarrow$ output of RL Model for $State_i$
9: ($Action_i$, $Penalty_{i+1}$) $\leftarrow$ CONSTRAINTSATISFACTIONMODULE ($probabilityMap$)
10: Allocate new tasks and migrate existing tasks based on $Action_i$
11: Execute tasks in edge-cloud infrastructure for interval SI_i
12: **end for**
End

算法（Algorithm 1）用于动态调度，主要步骤包括输入调度间隔数 N 和批量大小 B。对于每个调度间隔索引 i，从 1 到 N，如果 i > 1 且 i % B == 0，则在 RL 模型中进行反向传播。接着，预处理后的状态被发送到 RL 模型，获取概率图，使用约束满足模块（CSM）生成动作和惩罚，根据动作分配新任务并迁移现有任务，最后在边缘云基础设施上执行任务。

在复杂度分析方面，预处理输入状态的时间复杂度为 O(ab)，其中 a × b 是特征向量中的最大大小。生成动作和惩罚的时间复杂度为 $O(n^2)$，其中 n 是主机和任务的数量。因此，总的时间复杂度为 O(abN)，其中 N 是调度间隔数。

在具体步骤的时间复杂度分析中，预处理输入状态的时间复杂度为 O(ab)，发送预处理状态到 RL 模型的时间复杂度可以忽略不计，获取概率图的时间复杂度是常数时间，使用 CSM 生成动作和惩罚的时间复杂度为 $O(n^2)$，分配新任务和迁移现有任务的时间复杂度为 O(n)，执行任务的时间复杂度可以认为是常数时间操作。

论文的模型复现：

1. 使用：CNN+LSTM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torchvision.transforms as transform
from torch.autograd import Variable
import torch.nn.functional as F
```

```
from tqdm import tqdm
from sklearn import preprocessing
import os.path
import pickle
import math
import matplotlib.pyplot as plt
from sys import stdin, stdout
import io

torch.set_printoptions(threshold=10000)
np.set_printoptions(threshold=np.inf)


batch_size = 5
input_dim = 15
hidden_dim = 26
num_layers = 2
output_dim = 100

learning_rate = 0.00001

seq_dim = 1
PATH = './model1/'

no_of_hosts = 100
no_of_vms = 100


class DeepRL(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim, batch_size):
        super(DeepRL, self).__init__()

        file_path = PATH + 'running_model.pth'

        if not(os.path.isdir(PATH)):
            # print("here")
            os.mkdir(PATH)
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.batch_size = batch_size
        self.output_dim = output_dim
        self.hidden = []
        self.iter = 1
```

```
        self.loss_backprop = []
        self.loss_map = []

        for i in range(no_of_hosts):
            self.hidden += [self.init_hidden()]

        self.lstm = nn.LSTM(self.input_dim, self.hidden_dim, self.num_layers,
batch_first=True)
        # self.lstm2 = nn.LSTM(hidden_dim, hidden_dim, batch_first=True)
        self.conv1 = nn.Conv2d(1, 10, kernel_size=2)
        self.conv2 = nn.Conv2d(10, 1, kernel_size=3)
        self.conv3 = nn.Conv2d(10, 10, kernel_size=2)

        self.fc1 = nn.Linear(88, 1000)
        self.fc2 = nn.Linear(1000, 5000)
        self.fc3 = nn.Linear(5000, 10000)

        self.bn1 = nn.BatchNorm1d(26)

        if os.path.isfile(file_path):
            self.load_state_dict(torch.load(file_path))

            file = open(PATH + 'hidden_state.pickle','rb')
            self.hidden = pickle.load(file)

            file = open(PATH + 'loss_backprop.pickle','rb')
            self.loss_backprop = pickle.load(file)

            file = open(PATH + 'loss_map.pickle','rb')
            self.loss_map = pickle.load(file)

    def init_hidden(self):
        return (torch.zeros(self.num_layers,self.batch_size, self.hidden_dim),
                torch.zeros(self.num_layers,self.batch_size, self.hidden_dim))

    def forward(self, cnn_data, lstm_data):
        lstm_out_host = []
        for i in range(lstm_data.shape[1]):
            # print(lstm_data[:,i,:])
            out, hidden = self.lstm(lstm_data[:,i,:].view(-1,1,lstm_data.shape[2]))
            self.hidden[i] = hidden
            lstm_out_host += [out]
            #                                   lstm_out_host                                   +=
[self.bn1(out.view(-1,out.shape[2])).view(-1,1,out.shape[1])]
```

```
        # lstm_out = np.array(lstm_out)
        # print(lstm_out_host[0].shape)
        lstm_out = lstm_out_host[0]
        for i in range(1,lstm_data.shape[1]):
            lstm_out = torch.cat((lstm_out,lstm_out_host[i]),1)
        # print(lstm_out)
        # print(self.hidden[0].shape)
        x1 = lstm_out.reshape((lstm_out.shape[0],1,lstm_out.shape[1],lstm_out.shape[2]))
        x1 = F.relu(F.max_pool2d(self.conv1(x1),2))
        x1 = self.conv2(x1)

        cnn_data = cnn_data.reshape((-1,1,cnn_data.shape[1],cnn_data.shape[2]))
        x2 = F.relu(F.max_pool2d(self.conv1(cnn_data),2))
        x2 = self.conv2(x2)

        x3 = torch.cat((x1,x2),2)

        x3 = self.conv1(x3)
        x3 = F.max_pool2d((x3),2)
        x3 = self.conv2(x3)

        x3 = x3.reshape(-1,x3.shape[2]*x3.shape[3])

        x4 = self.fc1(x3)
        x4 = self.fc2(x4)
        x4  =self.fc3(x4)

        x4 = x4.reshape(-1,self.output_dim,self.output_dim)
        # print(x4)
        x4 = F.softmax(x4, dim=2)
        # print(x4[0][0])

        return x4


    def setInput(self, cnn_data, lstm_data):
        # for name, param in self.named_parameters():
        #         if param.requires_grad:
        #             print(name, param.data)
        self.vm_map = []
        for i in range(cnn_data.shape[0]):
            self.vm_map += [cnn_data[i][0]]
```

```
        # file = open(PATH + 'vm_map.pickle','wb')
        # pickle.dump(self.vm_map, file)

        lstm_data = preprocessing.normalize(lstm_data)
        cnn_data = preprocessing.normalize(cnn_data)

        train_cnn = Variable(torch.from_numpy(cnn_data).type(torch.FloatTensor).view(1,cnn_data.shape[0],cnn_data.shape[1]))
        train_lstm = Variable(torch.from_numpy(lstm_data).type(torch.FloatTensor).view(1,lstm_data.shape[0],lstm_data.shape[1]))

        # print(train_lstm.shape)
        self.output = self.forward(train_cnn, train_lstm)
        self.output = self.output.view(self.output.shape[1],self.output.shape[2])

        file = open(PATH+"DLoutput.txt", "w+")
        file.write(str(self.output))
        file.close()

        plt.imshow(self.output.detach().numpy(),cmap='gray')
        plt.savefig(PATH + 'DLoutput.jpg')
        plt.close()
        # file = open(PATH + 'output.pickle','wb')
        # pickle.dump(self.output, file)
        # print(self.output)


    def backprop(self, loss_parameters):
        if self.iter == 1:
            return("Init Loss")
        optimizer = torch.optim.Adam(self.parameters(), lr=learning_rate)

        loss_value = loss_parameters[3]/1000000 + loss_parameters[7] + loss_parameters[8]
        loss_value = torch.Tensor(np.array(loss_value)).type(torch.FloatTensor)

        # file = open('output.pickle','rb')
        # self.output = pickle.load(file)

        loss = self.output.min()
        loss.data = loss_value
```

```
        loss.backward()

        #update parameters
        optimizer.step()

        if self.iter%10 == 0:
            torch.save(model.state_dict(), PATH + 'running_model.pth')

            file = open(PATH + 'hidden_state.pickle','wb')
            pickle.dump(self.hidden, file)

            file = open(PATH + 'loss_backprop.pickle','wb')
            pickle.dump(self.loss_backprop, file)

            file = open(PATH + 'loss_map.pickle','wb')
            pickle.dump(self.loss_map, file)

            plt.plot(self.loss_backprop)
            plt.savefig(PATH + 'loss_backprop.jpg')
            plt.close()

            plt.plot(self.loss_map)
            plt.savefig(PATH + 'loss_map.jpg')
            plt.close()


        self.iter += 1

        self.loss_backprop += [loss.item()]
        return str(loss.item())

    def host_rank(self, vm):
        # print(self.output.shape)

        # file = open('output.pickle','rb')
        # self.output = pickle.load(file)

        host_list = self.output.data[vm]
        # print(host_list)
        indices = np.flip(np.argsort(host_list))
        # print(indices)
        s = ''
        for index in indices:
```

```
            s += str(index) + ' '
        return s

    def migratableVMs(self):
        # file = open('output.pickle','rb')
        # self.output = pickle.load(file)

        # file = open('vm_map.pickle','rb')
        # self.vm_map = pickle.load(file)

        output_index = np.argmax(self.output.data, axis=1)

        migratableIndex = []
        for i in range(len(output_index)):
            if self.vm_map[i] != output_index[i].item():
                migratableIndex += [i]
        # print(migratableIndex)
        s = ''
        for index in migratableIndex:
            s += str(index) + ' '
        return s

    def sendMap(self, data):
        if self.iter == 1:
            self.iter +=1
            return "Init Loss"
        optimizer = torch.optim.Adam(self.parameters(), lr=learning_rate)

        vmMap = np.zeros((100,100), dtype=int)
        # print(vmMap.shape)

        # file = open('output.pickle','rb')
        # self.output = pickle.load(file)

        loss = 0
        for i in range(len(data)):
            # l = data[i].split()
            y = data[i][1]
            vmMap[i][y] = 1
            # print(self.output[i][y])
            loss -= torch.log(self.output[i][y])

        plt.imshow(vmMap,cmap='gray')
        plt.savefig(PATH + 'sendMap.jpg')
```

```
plt.close()

file = open(PATH+"sendMap.txt", "w+")
file.write(str(vmMap))
file.close()

file = open(PATH+"DLmap.txt", "w+")
for i in range(len(data)):
    host_list = self.output.data[i]
    index = np.flip(np.argsort(host_list))[0]
    file.write(str(i) + " " + str(index) + "\n")
file.close()

loss /= len(data)
# print(loss)
loss.backward()

#update parameters
optimizer.step()

if self.iter%5 == 0:
    torch.save(model.state_dict(), PATH + 'running_model.pth')

    file = open(PATH + 'hidden_state.pickle','wb')
    pickle.dump(self.hidden, file)

    file = open(PATH + 'loss_backprop.pickle','wb')
    pickle.dump(self.loss_backprop, file)

    file = open(PATH + 'loss_map.pickle','wb')
    pickle.dump(self.loss_map, file)

    # globalFile.writeline(str(len(self.loss_map)))
    # globalFile.flush()
    plt.plot(self.loss_backprop)
    plt.savefig(PATH + 'loss_backprop.jpg')
    plt.clf()

    plt.plot(self.loss_map)
    plt.savefig(PATH + 'loss_map.jpg')
    plt.clf()


self.iter += 1
```

```
        self.loss_map += [loss.item()]
        return str(loss.item())


if __name__ == '__main__':

    model = DeepRL(input_dim, hidden_dim, num_layers, output_dim, batch_size)

    # inp = "backprop,CurrentTime 300.1;LastTime 0.0;TimeDiff 300.1;TotalEnergy
105358.10624075294;NumVsEnded 1.0;AverageResponseTime
0.0;AverageMigrationTime 0.0;TotalCost 0.3317772222222221;SLAOverall NaN"
    # inp = "setInput,CNN data;1 2 3;4 5 6;LSTM data;7 8 9;1 2 3"
    # inp = "host_rank,4"
    inp = "sendMap,1 0;2 0;3 1;4 2;5 2;6 3"
    # inp = 'migratableVMs,'
    inp = []
    globalFile = open(PATH+"logs.txt", "a")

    while(True):
        while(True):
            line = stdin.readline()
            if "END" in line:
                break
            inp.append(line)
        if inp[0] == 'exit':
            break
        funcName = inp[0]
        data = inp[1:]
        inp = []

        if 'setInput' in funcName:
            file = open(PATH+"DLinput.txt", "w+")
            file.writelines(data)
            file.close()
            flag = 0
            cnn_data = np.zeros((100, 26), dtype=float)
            lstm_data = np.zeros((100, 21), dtype=float)
            cnn_count = 0
            lstm_count = 0
            for val in data:
                val = val.replace('false','0')
                val = val.replace('true','1')
                val = val.replace('NaN','0')
```

```
            x = val.split(' ')
            if x[0] == 'CNN':
                flag = 1
                continue

            elif x[0] == "LSTM":
                flag = 2
                continue

            if flag == 1:
                for i in range(len(x)):
                    cnn_data[cnn_count][i] = float(x[i])
                cnn_count += 1

            elif flag == 2:
                for i in range(len(x)):
                    lstm_data[lstm_count][i] = float(x[i])
                lstm_count += 1

        model.setInput(cnn_data, lstm_data)

    elif funcName == 'backprop':
        loss_data = []
        for val in data:
            val = val.replace('false','0')
            val = val.replace('true','1')
            val = val.replace('NaN','0')
            # print(val)
            val = val.split()
            loss_data += [float(val[1])]

        stdout.write(model.backprop(loss_data))
        stdout.flush()

    elif 'getSortedHost' in funcName:
        vm = int(data[0])
        stdout.write(model.host_rank(vm))
        stdout.flush()

    elif 'getVmsToMigrate' in funcName:
        stdout.write(model.migratableVMs())
        stdout.flush()

    elif 'sendMap' in funcName:
```

```
            file = open(PATH+"DLsendMap.txt", "w+")
            file.writelines(data)
            file.close()
            hostVmMap = []
            for val in data:
                val = val.split()
                l = [int(val[0]), int(val[1])]
                hostVmMap += [l]

            stdout.write(model.sendMap(hostVmMap)+"\n")
            stdout.flush()
```

结构概述：

初始化模型：

定义了 LSTM 和 CNN 的网络结构，并加载现有的模型文件和隐藏状态。

初始化一些关键变量，例如 hidden、loss_backprop 和 loss_map，以跟踪训练过程中的状态和损失值。

前向传播 (forward)：

对 LSTM 和 CNN 数据分别处理，最后将两部分结果合并。

使用全连接层进一步处理数据，并生成输出概率分布，最终通过 Softmax 函数归一化。

模型的输入设置 (setInput)：

接收 CNN 和 LSTM 的数据，将其标准化并输入到模型中。

生成输出概率矩阵，并保存为图像和文本文件，以便后续分析。

反向传播 (backprop)：

根据提供的损失参数，计算总损失并更新模型权重。

每隔一定的迭代次数，将模型和相关状态保存到文件。

功能扩展：

提供虚拟机到主机排序功能 (host_rank)。

检测可迁移的虚拟机 (migratableVMs)。

接收主机与虚拟机的映射，并计算损失 (sendMap)。

2. 不使用 LSTM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torchvision.transforms as transform
from torch.autograd import Variable
```

```
import torch.nn.functional as F
from tqdm import tqdm
from sklearn import preprocessing
import os.path
import pickle
import math
import matplotlib.pyplot as plt
from sys import stdin, stdout
import io

torch.set_printoptions(threshold=10000)
np.set_printoptions(threshold=np.inf)

torch.set_default_tensor_type('torch.cuda.FloatTensor')

# batch_size = 5
input_dim = 15
hidden_dim = 26
num_layers = 2
output_dim = 100

learning_rate = 0.00001

seq_dim = 1
PATH = './model2/'

no_of_hosts = 100
no_of_vms = 100


class DeepRL(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim, batch_size):
        super(DeepRL, self).__init__()

        file_path = PATH + 'running_model.pth'

        if not(os.path.isdir(PATH)):
            # print("here")
            os.mkdir(PATH)
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.batch_size = batch_size
        self.output_dim = output_dim
```

```
        self.hidden = []
        self.iter = 1
        self.loss_backprop = []
        self.loss_map = []
        # self.scheduler = lr_scheduler.CosineAnnealingLR(optimizer, 5*24*12,
eta_min=learning_rate)

        for i in range(no_of_hosts):
            self.hidden += [self.init_hidden()]

        # self.lstm = nn.LSTM(self.input_dim, self.hidden_dim, self.num_layers,
batch_first=True)
        # # self.lstm2 = nn.LSTM(hidden_dim, hidden_dim, batch_first=True)
        # self.conv1 = nn.Conv2d(1, 10, kernel_size=2)
        # self.conv2 = nn.Conv2d(10, 1, kernel_size=3)
        # self.conv3 = nn.Conv2d(10, 10, kernel_size=2)

        # self.fc1 = nn.Linear(88, 1000)
        # self.fc2 = nn.Linear(1000, 5000)
        # self.fc3 = nn.Linear(5000, 10000)

        # self.bn1 = nn.BatchNorm1d(26)

        self.relu = nn.ELU()

        self.fc1 = nn.Linear(24200, 5000)
        self.fc2 = nn.Linear(5000, 2500)
        self.fc3 = nn.Linear(2500, 2500)
        self.fc4 = nn.Linear(2500, 2500)
        self.fc5 = nn.Linear(2500, 2500)
        self.fc6 = nn.Linear(2500, 10000)

        if os.path.isfile(file_path):
            self.load_state_dict(torch.load(file_path))

            file = open(PATH + 'hidden_state.pickle','rb')
            self.hidden = pickle.load(file)

            file = open(PATH + 'loss_backprop.pickle','rb')
            self.loss_backprop = pickle.load(file)

            file = open(PATH + 'loss_map.pickle','rb')
            self.loss_map = pickle.load(file)
```

```
def init_hidden(self):
    return (torch.zeros(self.num_layers,self.batch_size, self.hidden_dim),
            torch.zeros(self.num_layers,self.batch_size, self.hidden_dim))

def forward(self, cnn_data, lstm_data):
    cnn_data = cnn_data.reshape(-1,cnn_data.shape[1]*cnn_data.shape[2])
    lstm_data = lstm_data.reshape(-1,lstm_data.shape[1]*lstm_data.shape[2])
    data = torch.cat((cnn_data, lstm_data),1).cuda()

    data = self.relu(self.fc1(data))
    data = self.relu(self.fc2(data))
    data = self.relu(self.fc3(data))
    data = self.relu(self.fc4(data))
    data = self.relu(self.fc5(data))
    data = self.relu(self.fc6(data))

    data = data.reshape(-1,self.output_dim,self.output_dim)
    data = F.softmax(data, dim=2)

    return data

def setInput(self, cnn_data, lstm_data):
    # for name, param in self.named_parameters():
    #     if param.requires_grad:
    #         print(name, param.data)
    self.vm_map = []
    for i in range(cnn_data.shape[1]):
        self.vm_map += [cnn_data[0][i][0]]

    # file = open(PATH + 'vm_map.pickle','wb')
    # pickle.dump(self.vm_map, file)

    # lstm_data = preprocessing.normalize(lstm_data)
    # cnn_data = preprocessing.normalize(cnn_data)

    train_cnn  = Variable(torch.from_numpy(cnn_data).type(torch.FloatTensor))
    train_lstm = Variable(torch.from_numpy(lstm_data).type(torch.FloatTensor))

    train_cnn.cuda()
    train_lstm.cuda()
    # print(train_lstm.shape)
    self.output = self.forward(train_cnn, train_lstm)
    # self.output = self.output.view(self.output.shape[1],self.output.shape[2])
```

```
        for out in self.output:
            file = open(PATH+"DLoutput.txt", "w+")
            file.write(str(out))
            file.close()

            plt.imshow(out.cpu().detach().numpy(),cmap='gray')
            plt.savefig(PATH + 'DLoutput.jpg')
            plt.close()
        # file = open(PATH + 'output.pickle','wb')
        # pickle.dump(self.output, file)
        # print(self.output)


    def backprop(self, loss_parameters):
        if self.iter == 1:
            return("Init Loss")

        loss_value = loss_parameters[3]/1000000 + loss_parameters[7] +
loss_parameters[8]
        loss_value = torch.Tensor(np.array(loss_value)).type(torch.FloatTensor)

        # file = open('output.pickle','rb')
        # self.output = pickle.load(file)

        loss = self.output.min()
        loss.data = loss_value

        loss.backward()

        #update parameters
        optimizer.step()

        if self.iter%1 == 0:
            torch.save(model.state_dict(), PATH + 'running_model.pth')

            file = open(PATH + 'hidden_state.pickle','wb')
            pickle.dump(self.hidden, file)

            file = open(PATH + 'loss_backprop.pickle','wb')
            pickle.dump(self.loss_backprop, file)

            file = open(PATH + 'loss_map.pickle','wb')
            pickle.dump(self.loss_map, file)
```

```
            plt.plot(self.loss_backprop)
            plt.savefig(PATH + 'loss_backprop.jpg')
            plt.close()

            plt.plot(self.loss_map)
            plt.savefig(PATH + 'loss_map.jpg')
            plt.close()


        self.iter += 1

        self.loss_backprop += [loss.item()]
        return str(loss.item())

    def host_rank(self, vm):
        # print(self.output.shape)

        # file = open('output.pickle','rb')
        # self.output = pickle.load(file)

        host_list = self.output.data[vm]
        # print(host_list)
        indices = np.flip(np.argsort(host_list))
        # print(indices)
        s = ''
        for index in indices:
            s += str(index) + ' '
        return s

    def migratableVMs(self):
        # file = open('output.pickle','rb')
        # self.output = pickle.load(file)

        # file = open('vm_map.pickle','rb')
        # self.vm_map = pickle.load(file)

        output_index = np.argmax(self.output.data, axis=1)

        migratableIndex = []
        for i in range(len(output_index)):
            if self.vm_map[i] != output_index[i].item():
                migratableIndex += [i]
        # print(migratableIndex)
        s = ''
```

```
        for index in migratableIndex:
            s += str(index) + ' '
        return s

    def sendMap(self, data_input):

        total_loss = 0
        index = 0
        for data in data_input:
            vmMap = np.zeros((100,100), dtype=int)
            # print(vmMap.shape)

            # file = open('output.pickle','rb')
            # self.output = pickle.load(file)

            loss = 0
            for i in range(len(data)):
                # l = data[i].split()
                y = data[i][1]
                vmMap[i][y] = 1
                # print(self.output[i][y])
                loss -= torch.log(self.output[index][i][y])

            plt.imshow(vmMap,cmap='gray')
            plt.savefig(PATH + 'sendMap.jpg')
            plt.close()

            file = open(PATH+"sendMap.txt", "w+")
            file.write(str(vmMap))
            file.close()
            index += 1
            loss /= len(data)
            total_loss += loss

        total_loss /= len(data_input)
        # print(loss)
        total_loss.cuda()
        total_loss.backward()

        #update parameters
        optimizer.step()

        if self.iter%1 == 0:
            torch.save(model.state_dict(), PATH + 'running_model.pth')
```

```
                file = open(PATH + 'hidden_state.pickle','wb')
                pickle.dump(self.hidden, file)

                file = open(PATH + 'loss_backprop.pickle','wb')
                pickle.dump(self.loss_backprop, file)

                file = open(PATH + 'loss_map.pickle','wb')
                pickle.dump(self.loss_map, file)

                # globalFile.writeline(str(len(self.loss_map)))
                # globalFile.flush()
                plt.plot(self.loss_backprop)
                plt.savefig(PATH + 'loss_backprop.jpg')
                plt.clf()

                plt.plot(self.loss_map)
                plt.savefig(PATH + 'loss_map.jpg')
                plt.clf()


            self.iter += 1

            self.loss_map += [total_loss.item()]
            return str(total_loss.item())


def preprocess(data, mean_old, std_old, flag):
    alpha = 0.5
    beta = 0.5
    for i in range(data.shape[2]):
        l = data[:,:,i]
        mean_new = np.mean(l)
        std_new = np.std(l)
        if flag == 0:
            mean_new = alpha*mean_new + (1-alpha)*mean_old
            std_new = beta*std_new + (1-beta)*std_old
        if std_new!=0:
            data[:,:,i] = (data[:,:,i] - mean_new) / std_new
        else:
            data[:,:,i] = 0
    return (data, mean_new, std_new)

def normalize(data, min_max):
```

```
    for i in range(data.shape[2]):
        if min_max[i][1] == min_max[i][0]:
            data[:,:,i] = 0
        else:
            data[:,:,i] = (data[:,:,i] - min_max[i][0]) / (min_max[i][1] -
min_max[i][0])
    return data


if __name__ == '__main__':
    global optimizer

    file = open('../Deep-Learning/cnn_min_max.pickle','rb')
    cnn_min_max = pickle.load(file)

    file = open('../Deep-Learning/lstm_min_max.pickle','rb')
    lstm_min_max = pickle.load(file)

    batch_size = 12
    model = DeepRL(input_dim, hidden_dim, num_layers, output_dim, batch_size)
    model.cuda()
    # inp = "backprop,CurrentTime 300.1;LastTime 0.0;TimeDiff 300.1;TotalEnergy
105358.10624075294;NumVsEnded 1.0;AverageResponseTime
0.0;AverageMigrationTime 0.0;TotalCost 0.3317772222222221;SLAOverall NaN"
    # inp = "setInput,CNN data;1 2 3;4 5 6;LSTM data;7 8 9;1 2 3"
    # inp = "host_rank,4"
    inp = "sendMap,1 0;2 0;3 1;4 2;5 2;6 3"
    # inp = 'migratableVMs,'
    inp = []
    globalFile = open(PATH+"logs.txt", "a")

    batch_count_forward = 0
    batch_count_backward = 0
    cnn_input = []
    lstm_input = []
    hostVm_input = []
    mean = 0
    std = 0
    data_flag = 0

    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    # device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    # print(device)
```

```
while(True):
    while(True):
        line = stdin.readline()
        if "END" in line:
            break
        inp.append(line)
    if inp[0] == 'exit':
        break
    funcName = inp[0]
    data = inp[1:]
    inp = []

    if 'setInput' in funcName:
        file = open(PATH+"DLinput.txt", "w+")
        file.writelines(data)
        file.close()
        flag = 0
        cnn_data = np.zeros((100, 126), dtype=float)
        lstm_data = np.zeros((100, 116), dtype=float)
        cnn_count = 0
        lstm_count = 0
        for val in data:
            val = val.replace('false','0')
            val = val.replace('true','1')
            val = val.replace('NaN','0')
            x = val.split(' ')
            if x[0] == 'CNN':
                flag = 1
                continue

            elif x[0] == "LSTM":
                flag = 2
                continue

            if flag == 1:
                for i in range(len(x)):
                    cnn_data[cnn_count][i] = float(x[i])
                cnn_count += 1

            elif flag == 2:
                for i in range(len(x)):
                    lstm_data[lstm_count][i] = float(x[i])
                lstm_count += 1
```

```
        # cnn_data = preprocessing.normalize(cnn_data)
        # lstm_data = preprocessing.normalize(lstm_data)

        cnn_input += [cnn_data]
        lstm_input += [lstm_data]
        batch_count_forward += 1

        if batch_count_forward == batch_size:
            cnn_data = np.array(cnn_input)
            lstm_data = np.array(lstm_input)

            # cnn_data, mean, std = preprocess(cnn_data,mean,std,data_flag)
            # lstm_data, mean, std = preprocess(lstm_data,mean,std,data_flag)
            # data_flag = 1

            cnn_data = normalize(cnn_data, cnn_min_max)
            lstm_data = normalize(lstm_data, lstm_min_max)

            # print(cnn_data.shape, lstm_data.shape)
            model.setInput(cnn_data, lstm_data)
            cnn_input = []
            lstm_input = []
            batch_count_forward = 0

    elif funcName == 'backprop':
        loss_data = []
        for val in data:
            val = val.replace('false','0')
            val = val.replace('true','1')
            val = val.replace('NaN','0')
            # print(val)
            val = val.split()
            loss_data += [float(val[1])]

        stdout.write(model.backprop(loss_data))
        stdout.flush()

    elif 'getSortedHost' in funcName:
        vm = int(data[0])
        stdout.write(model.host_rank(vm))
        stdout.flush()

    elif 'getVmsToMigrate' in funcName:
```

```
        stdout.write(model.migratableVMs())
        stdout.flush()

    elif 'sendMap' in funcName:
        file = open(PATH+"DLsendMap.txt", "w+")
        file.writelines(data)
        file.close()
        if model.iter == 1:
            stdout.write("Init Loss\n")
            stdout.flush()
            model.iter += 1
            continue

        hostVmMap = []
        for val in data:
            val = val.split()
            l = [int(val[0]), int(val[1])]
            hostVmMap += [l]

        hostVm_input += [hostVmMap]

        batch_count_backward += 1
        # print(hostVmMap)
        if batch_count_backward == batch_size:
            # print(model.sendMap(hostVm_input))
            # print(hostVm_input)
            ans = model.sendMap(hostVm_input)
            # file1 = open('check.txt','a')
            # file1.write(ans + '\n')
            # file1.close()
            # print(batch_count_forward, batch_count_backward)
            stdout.write(ans+"\n")
            stdout.flush()
            hostVm_input = []
            batch_count_backward = 0

        else:
            stdout.write(str(0.1)+"\n")
            stdout.flush()
```

1. 模型定义

输入结构： CNN（卷积神经网络）和 LSTM（长短时记忆网络）数据分别作为输入。

数据在输入时通过线性层和激活函数（ELU）处理。

网络结构： 多层全连接层（fc1 至 fc6）用于特征提取和输出生成。
使用 F.softmax 函数对最后一层进行归一化，输出维度为[output_dim, output_dim]。
隐藏状态： 每个主机初始化隐藏状态，用于存储训练过程中的中间状态。

2. 数据处理

预处理： preprocess 和 normalize 函数对数据进行标准化处理，减小均值和标准差对数据分布的影响。
min-max 归一化在归一化范围不变的情况下将数据标准化。
输入格式： 将 CNN 和 LSTM 数据分别转化为固定维度矩阵，拼接后作为神经网络输入。

3. 核心功能

前向传播（Forward Pass）： 调用 forward 函数将输入数据经过多层网络，生成主机-虚拟机的映射概率分布。
反向传播（Backpropagation）： 根据自定义的损失函数，计算梯度并优化网络参数。
保存中间训练状态，包括模型权重、隐藏状态和损失值。
虚拟机迁移： 使用 migratableVMs 函数判断哪些虚拟机可以迁移。
计算每台虚拟机最优主机分配的排名（host_rank）。
映射生成： sendMap 函数生成主机-虚拟机的最终分配图，并保存为图片和文本。

4. 文件交互

使用 Pickle 序列化工具保存和加载模型状态及中间变量（如隐藏状态和损失）。
在特定步骤将训练结果保存为图像（如损失曲线和输出分布）。

实验输出记录：

CNN number of VMs 11

```
1 466.1663611111111 466.1663611111111 466.1663611111111 0.0 2500 0 4000 4000 365 10000 10000 26 100000 0 1997848 false NaN 0 466.1663611111111 4853.833638888889 511639 99990000 0 1 105358.10624075294
1 0.0 0.0 0.0 0.0 2500 0 4000 4000 0 10000 10000 0 100000 0 0 false NaN 2 0.0 5320.0 511639 99990000 0 1 0.0
1 0.0 0.0 0.0 0.0 2500 0 4000 4000 0 10000 10000 0 100000 0 0 false NaN 3 0.0 5320.0 511639 99990000 0 1 0.0
1 0.0 0.0 0.0 0.0 2500 0 4000 4000 0 10000 10000 0 100000 0 0 false NaN 4 0.0 5320.0 511639 99990000 0 1 0.0
1 0.0 0.0 0.0 0.0 2500 0 4000 4000 0 10000 10000 0 100000 0 0 false NaN 5 0.0 5320.0 511639 99990000 0 1 0.0
1 0.0 0.0 0.0 0.0 2500 0 4000 4000 0 10000 10000 0 100000 0 0 false NaN 6 0.0 5320.0 511639 99990000 0 1 0.0
2 0.0 0.0 0.0 0.0 2500 0 8000 8000 0 10000 10000 0 100000 0 0 false NaN 7 0.0 5320.0 507639 99990000 0 1 0.0
```

2 0.0 0.0 0.0 0.0 2500 0 8000 8000 0 10000 10000 0 100000
0 0 false NaN 8 0.0 5320.0 507639 99990000 0 1 0.0
2 0.0 0.0 0.0 0.0 2500 0 8000 8000 0 10000 10000 0 100000
0 0 false NaN 9 0.0 5320.0 507639 99990000 0 1 0.0
2 0.0 0.0 0.0 0.0 2500 0 8000 8000 0 10000 10000 0 100000
0 0 false NaN 10 0.0 5320.0 507639 99990000 0 1 0.0
2 0.0 0.0 0.0 0.0 2500 0 8000 8000 0 10000 10000 0 100000
0 0 false NaN 11 0.0 5320.0 507639 99990000 0 1 0.0
LSTM
105358.10624075294 351.0766619152047 1229.0 0.0033177722222222225
NaN 0.0876252558479532 0.17525051169590647 2 511639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 511639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 511639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 511639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 511639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 507639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 507639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 507639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 507639 515639
99990000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0

0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0

0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0

0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0
0.0 210.0 1229.0 0.0033177722222222225 NaN 0.0 0.0 2 515639 515639
100000000 100000000 0 0 0

0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0
0.0 210.0 1229.0 0.003317772222222225 NaN 0.0 0.0 2 515639 515639 100000000 100000000 0 0 0