上海大学
SHANGHAI UNIVERSITY

本科毕业论文（设计）

UNDERGRADUATE THESIS (PROJECT)

题　　目：基于强化学习的自适应物联网流处理系统设计与实现

学　　院：计算机工程与科学学院

专　　业：网络空间安全

学　　号：19121518

学生姓名：吴裕恒

指导教师：曹晨红

起讫日期：2023.02.06-2023.06.02

姓　　名：吴裕恒　　学号：19121518

论文题目：基于强化学习的自适应物联网流处理系统设计与实现

原 创 性 声 明

本人声明：所呈交的论文是本人在指导教师指导下进行的研究工作。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已发表或撰写过的研究成果。参与同一工作的其他同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名：吴裕恒 日 期：2023.05.25

本论文使用授权说明

本人完全了解上海大学有关保留、使用学位论文的规定，即：学校有权保留论文及送交论文复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容。

（**保密的论文在解密后应遵守此规定**）

签 名：吴裕恒 指导教师签名：＿＿＿＿ 日期：2023.05.25

目 录

基于强化学习的物联网自适应流处理系统的设计与实现

摘 要

近年来随着 AI，5G 以及嵌入式技术的发展，物联网技术与应用也产生了巨大的变革。再加上近年来万物互联的理念普及，车联网，智能家居，智慧城市，智慧工业等应用的发展和推进，物联网中接入设备以及产生的数据量激增，传统的基于云计算的数据流处理模式难以满足对于低时延和高体验质量的需求。为了降低时延并提高体验质量，在云计算的基础之上，物联网边缘计算模式开始受到关注。通过在靠近物或数据源头的一侧，采用网络、计算、存储、应用核心能力为一体的开放平台，就近提供最近端服务，边缘计算将处理能力转移到更靠近用户和设备的位置，显著提高了应用程序性能，降低了带宽需求，并在提供了更快的实时性的同时利用边缘侧更加充足的计算资源提高了体验质量。然而，尽管物联网边缘计算模式已经显著改善了低时延和体验质量遇到的问题，边缘计算模式仍然面临着网络资源的稀缺性和高动态性的挑战。此外，由于物联网中的不同应用所产生的数据流的处理需求存在差异性，传统的通过开发人员手动配置的边缘计算的策略可能缺乏对数据流内容以及网络资源动态性的合理度量，往往导致所配置的策略仅仅为次优并且难以应对复杂多变的网络环境。为了优化这一问题，近期出现一些工作[1-3]通过引入基于深度强化学习的方法来学习一个更鲁棒的策略，取得了较好的效果。但是这些工作仍然存在一定的局限性。一方面，这些工作往往只限于某一个特定的场景，难以适配到不同类型的数据流处理任务中。另一方面，一部分工作[1-2]仅仅将场景限制在只有一个边缘服务器中，过度简化了边缘环境，难以反映实际的资源情况，限制了可用于处理数据流的计算资源。本文为了解决以上两个问题，基于深度强化学习算法，设计了一个能够自适应处理不同物联网数据流的系统。用户通过指定优化目标和配置集合，系统即可以适配不同类型的数据流和优化目标。通过以物联网中视频流处理任务作为案例进行实现与测试，我们发现该自适应系统相较于手动配置的策略能够满足用户对时延和体验质量的需求。

关键词：物联网，边缘计算，深度强化学习

The Design and Implementation of Configuration-Adaptive Iot Streaming with Deep Reinforcement Learning

ABSTRACT

In recent years, with the development of AI, 5G, and embedded technology, the Internet of Things (IoT) and applications have undergone significant changes. In addition, the concept of the interconnectedness of all things has become popular in recent years, which has led to the development and promotion of applications such as connected cars, smart homes, smart cities, and intelligent industries. As a result, the amount of data generated by IoT devices has increased dramatically, making it difficult for traditional cloud computing-based data flow processing models to meet the demands of low latency and high quality of experience.

To address these issues, edge computing has drawn attention as a complementary solution to cloud computing. By using an open platform that integrates network, computing, storage, and application core capabilities on the side of the device or data source, edge computing provides nearby services to users and devices, thereby significantly improving application performance, reducing bandwidth requirements, and providing faster real-time processing by using the abundant computing resources available at the edge. Although the edge computing model has significantly improved the problem of low latency and quality of experience, it still faces challenges such as the scarcity and high dynamics of network resources.

In addition, the processing requirements of data flows generated by different IoT applications vary, and traditional edge computing strategies manually configured by researchers may lack reasonable metrics for the content of data flows and the dynamics of network resources, often resulting in suboptimal strategies that are difficult to adapt to complex and changing network environments. Previous works[1-3] have introduced

deep reinforcement learning-based methods to optimize strategies with good results. However, these works still have limitations. On the one hand, they are often limited to specific scenarios and difficult to adapt to other types of data flow processing tasks. On the other hand, some works only consider a single edge server, limiting the available computing resources for processing data flows.

To address these limitations, this paper proposes a system based on deep reinforcement learning algorithms that can adaptively handle different IoT data streams. By allowing users to specify optimization objectives and configuration sets, the system can adapt to different types of data streams and optimization goals. Through testing with video stream processing tasks in the IoT domain, we found that this adaptive system can meet users' requirements for latency and quality of experience compared to manually-configured.

Keywords: Internet of Things, Edge Computing, Deep Reinforcement Learning

1 引言

1.1 研究背景及意义

物联网（Internet of Things, IoT）是一种新型的信息技术，它将传感器、智能设备、云计算等技术相互融合，实现了物理世界和数字世界的连接和互通，使得各种设备和系统之间可以进行信息共享和协同工作。IDC 发布的 2022 年 V1 版 IDC《全球物联网支出指南》（IDC Worldwide Internet of Things Spending Guide）中指出 2021 年全球物联网（企业级）支出规模达 6,902.6 亿美元，并有望在 2026 年达到 1.1 万亿美元，五年（2022-2026）复合增长率 10.7%。其中，中国企业级市场规模将在 2026 年达到 2,940 亿美元，复合增长率 13.2%。全球占比约为 25.7%，继续保持全球最大物联网市场体量。物联网已经成为我国发展战略的重点发展方向，是“中国制造 2025”和“十三五规划”的重要组成部分。

随着万物互联的趋势不断发展以及各种物联网设备的普及，物联网技术已经在众多领域得到了广泛应用，如智慧城市、智能家居、智能交通、智能制造等。以感知设备为代表的各类物联网的设备和新应用涌现出来，其中相当一部分物联网关键应用需要对大量接入的物联网设备产生的海量实时数据进行低延时，高吞吐的流式处理。例如，北京和伦敦已经在城市建设中部署了数百万的监控摄像头用于城市监控和交通调节，通过对交通信号灯，车辆，行人等交通要素的实时监测，通过分析及合理规划，提高了交通效率和安全性。在智能家居场景中，物联网技术可以实现家居设备的互联互通和远程控制，为人们带来更便利舒适的生活环境。在新型智慧城市的建设中，大量传感器部署在城市中，收集温度，空气质量，湿度等信息，来指导人们的生活，提高了生活舒适度。

尽管物联网技术已经为人们的生活带来巨大的改善，越来越复杂的应用场景和人们日益增长的需求仍给物联网技术带来严峻的挑战。一方面，由于接入设备数量的激增，物联网设备产生的数据量急速增加。IDC 的报告中指出 2020 年全球的创造了大约 64ZB 的数据。到 2025 年，这一数字有望达到 163ZB，其中物联网设备生成的数据量预计将达到 73.1ZB，占全球数据总量的 44%。以监控摄像头为例，我国在 2022 年部署的视频监控摄像头达到 27.6 亿台，每天每台 1080p 分辨率的摄像头产

生的数据量是 64G，一台设备每年的数据量是 23T。由此可见，物联网应用中每时每刻都在产生大量的实时数据，大量的实时数据传输所带来的巨大负载加剧了网络带宽资源的稀缺问题。另一方面，大量的物联网设备是通过无线网络连接到边缘设备和因特网中，无线网络的带宽不仅稀缺还更易受到环境的影响，不同的信号传输距离，传输环境，传输强度以及网络负载都会对无线网络带宽产生影响，使得无线网络的带宽动态性非常高。此外，多终端、多应用任务的共存还将进一步加剧网络资源的动态性。在物联网场景下，局域网中往往存在多个 IoT 设备与应用。例如，在一个工厂中存在多个 IoT 设备和应用程序（机器人、传感器和计算机控制系统等），这些设备和应用程序需要协同工作以实现工厂的自动化生产和监测。由于设备的动态加入和离开，网络资源的变化也会变得更加复杂。此外，物联网中的应用虽然提供的服务和面向的场景不同，但是基本都具有“一特点和两要求”：

- 计算密集型特点：海量实时数据的处理分析以及机器学习方法的使用需要消耗大量计算资源。
- 低时延要求：对实时数据的处理分析结果往往会提供给后续的其他决策任务，分析结果实时性是后续一系列处理分析可靠性的重要保证。例如，对无人驾驶场景中互联设备产生的数据做实时分析，分析结果可以规划无人驾驶的汽车的下一步行为，如果分析结果并不是实时得到的，就有发生事故的风险。
- 高体验质量（Quality of Experience，QoE）要求：不同的应用和服务所产生的数据流种类不同，不同类型数据流的处理方式也存在差异，再加上个性化的用户需求，要想得到高体验质量就需要对针对不同数据流任务的处理策略进行优化。

在网络资源稀缺和受限的条件下，平衡低时延和高体验质量的需求变得非常困难，物联网任务的计算密集性和物联网设备计算资源稀缺的冲突也给这一目标带来更大的挑战。在早期的应用中，由于物联网设备的计算能力受限。较为成熟的解决方案是使用云计算，通过将采集的实时数据上传到计算资源丰富的云数据中心，进行集中处理，有效地缓解了本地设备计算资源稀缺和用户高体验质量的问题。但是由于物联网设备的地域性和无线网络的动态性，上传至固定的数据中心会不可避免地产生较高的时延。随着近年来人们对于用户体验质量（QoE）和个人数据隐私性的要求日益提高。传统的云计算模式难以满足用户的低时延需求。此外，数据上传至数据中心也会带来一定的隐私问题。传统流处理方法也缺乏对网络资源状况的可见性，

且缺少对不同应用的体验质量需求差异性的考虑，从而难以适应动态的网络环境。

近年来，随着基于 RISC 和 ARM 的低功耗集成芯片的技术进步和成熟以及 GPU 等计算设备的算力提升，物联网应用的服务提供商在网络中部署的服务器也开始具有一定的计算资源用于流式数据处理任务。在此基础上，部分工作[1-2]提出的边缘计算模式开始受到关注。边缘计算通过将部分计算任务从物联网设备卸载到较近的具有计算能力的边缘服务器上，从而有效降低了数据上传至更远距离的云数据中心所产生的时延，提升了实时性和用户体验质量。通过有效规划任务卸载，综合利用物联网设备，边缘服务器和云计算中心的资源，在网络资源稀缺的限制下，有效改善了低时延和高体验质量之间的平衡。物联网边缘计算架构如图1.1所示。

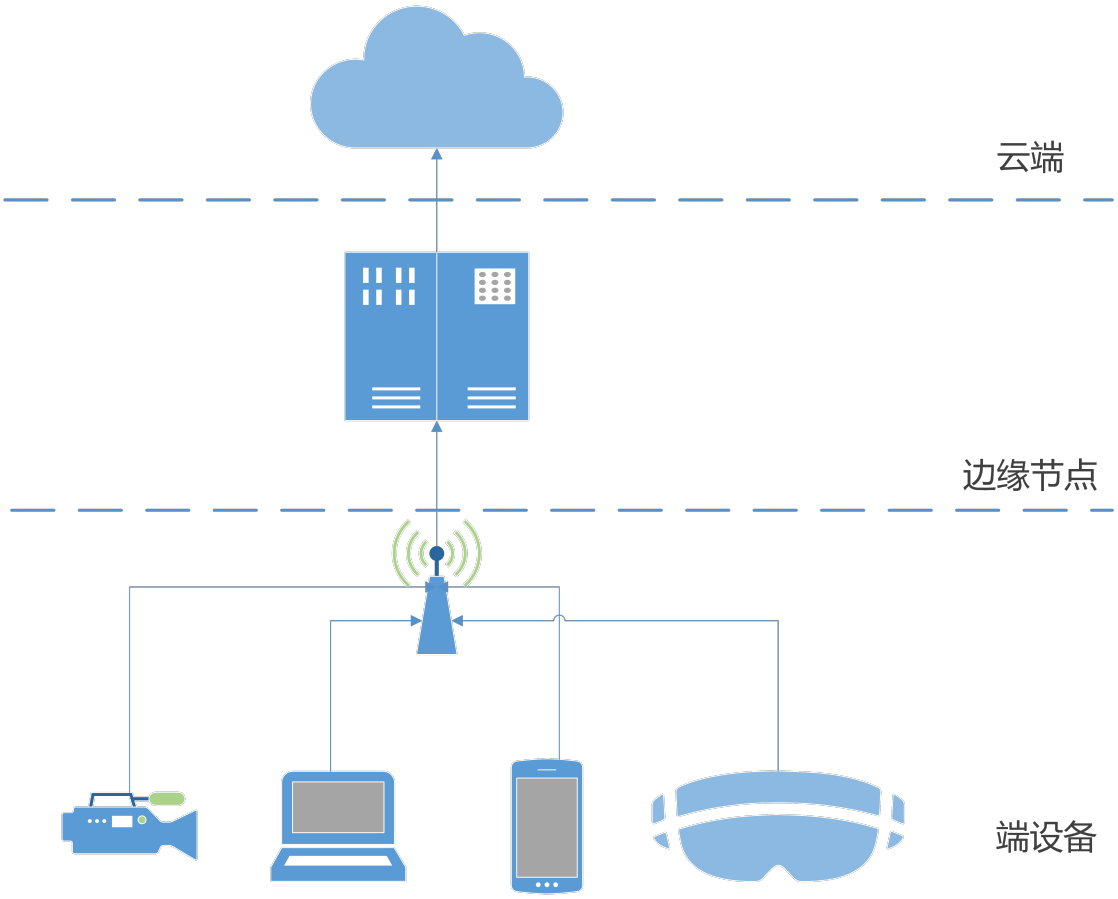

图 1.1　物联网边缘计算体系架构

尽管边缘计算很大程度提高了实时性和用户体验，在边缘计算架构下，物联网数据流处理依然面临着网络资源稀缺性的挑战。与云数据中心相比，在边缘端执行数据流处理的应用在资源可用性方面有更多的限制，包括带宽资源，计算资源等。由于无线网络的高度动态性，物联网中网络资源的感知存在以下挑战：

（1）如何精确、高效地感知动态网络资源。由于大量的物联网设备通过无线通信技术接入网络，使得网络路由拓扑和可用资源不断变化，加剧了网络资源的动态性。此外，网络中的计算设备呈现多样化的趋势，不同的计算设备在通信、计算、能耗方面也存在着巨大差异。同时，随着任务的执行和推进，任务对计算设备软硬件资源的需求以及网络条件都在不断变化，无线带宽的波动甚至可以达到秒级。因此，如何准确感知和预测动态网络资源的情况，如网络路由拓扑、可用带宽信息、网络计算资源等，为任务的实时协同提供基础和理论依据，具有很高的挑战性和研究价值。

在获得了网络资源信息之后，另一个挑战是（2）如何高效、自适应地协同处理多个物联网流处理应用。在物联网中，不同应用具有不同的性能需求，如精度、延迟和能耗等方面的要求，同时，感知数据不同维度的质量也会影响应用的性能。在网络资源条件难以满足所有维度数据都保持高质量的情况下，可以通过适当牺牲数据次要维度的质量，以降低对网络资源的消耗，最大程度地避免对关键性能的影响。已有方法通过编写手动策略[4]来进行协调，例如指定在带宽不足时，将视频流的帧率固定的下调一定的比例。然而，这种手动制定策略需要开发人员具备深入的专业知识同时需要对应用场景的性能瓶颈进行深入分析。然而，研究人员制定的基于规则的策略缺乏对数据质量和性能结果的定量度量，往往会导致策略次优且难以适应高度动态的网络。更严重的问题是，不同的应用对关键性能需求存在差异，为一个应用配置的策略可能不适用于其他应用。例如，对于注重用户体验质量（QoE）的直播视频流应用来说，在带宽受限时，应当配置策略以降低分辨率而维持较高的帧率，以保持连贯的视频流体验。然而，对于面部检测的视频流应用来说，降低分辨率会导致依赖图像细节的分析结果精度下降。因此，如何让开发人员能够快速制定多个应用的协同策略，并能够根据网络资源状况自适应地选择最优策略，以实现多应用之间实时高效的协同是一个关键挑战。

因此，如何在网络资源受到限制的条件下，为不同物联网数据流处理应用制定自适应的处理策略，实现降低时延的同时提高体验质量，将为物联网的技术发展和应用落地做出巨大贡献，也是本文重点讨论和解决的问题。

1.2 研究现状

近年来针对以上挑战，领域内涌现了大量相关的工作。一些广泛使用的流数据处理系统 Spark Streaming[5]，Apache Flink[6]等并未对网络资源状况问题进行考

虑。近期一些工作[2,4,7]主要将通过将数据流卸载问题转化为一个优化问题，以 Awstream[4]为例，通过使用经过离线训练的得到的帕累托最优化策略来筛选出备选的配置，根据实时测量的带宽从给出的备选配置中选择合适的数据传输率配置，从而降低对带宽的需求。JCAB[2]在多客户端数据卸载到单一边缘服务器的场景中，通过使用李雅普诺夫优化将原始的时间平均（time-averaged）问题转化为一个时序的最小化问题，从而实现只需要根据当前时刻的网络信息就可以更新迭代自适应卸载策略。部分工作针对特定的任务场景进行了优化。例如，以物联网视频流处理为例，VideoStorm[7]在大型集群中构建了一个为不同视频资源分配不同计算资源的策略。ziehen 等人针对 VR 视频应用场景提出了 EdgeVR[8]，着重缓解了 VR 视频数据处理的能耗问题。但是随着物联网场景中的接入设备激增，设备异构性和网络的高动态性等问题使得基于优化的方法往往获得的是次优的策略。此外，基于优化的方法需要很依赖对环境的分析建模以及优化目标的定量度量，这一过程需要大量的专家知识和繁复的数学推导，实现较为困难。且一旦任务和优化目标发生变化，已经获得的策略很难适配到其他任务中。

在 2016 年 AlphaGo 在围棋比赛中战胜围棋顶尖选手，深度强化学习方法在序列决策问题中的优异性能和鲁棒性受到关注。深度强化学习的方法也开始引入到数据流处理问题中，Pensieve[3]首先将深度强化学习的方法引入到 DASH（Dynamic Adaptive Streaming over HTTP）[9]协议的 ABR（Adaptive Bitrate Streaming）算法中，作者通过在构建的一个简单的仿真环境中模拟基于 DASH 的视频流传输过程训练了一个 ABR 策略，并将训练好的策略在真实环境中测试，取得了很好的效果，显示出了深度强化学习在解决复杂动态网络环境场景中序列决策问题的潜力。CASVA[1]在单客户端单边缘服务器场景下实时视频流卸载和分析任务中，使用强化学习算法生成卸载决策，只需对网络环境当前带宽以及其他客户端状态信息的观测即获得了较好的卸载策略，在有效降低视频流传输时延的情况下，维持了较好的视频流分析精确度。

1.3 研究内容

虽然1.2中提到的工作在物联网边缘计算模式之上极大程度的改善了低时延和体验质量的问题，但是这些工作仍然存在一定的局限性。(1) 以 Pensieve[3]，CASVA[1]等为代表的工作往往只限于特定的视频流任务处理场景，难以适配到其他类型的数据

流处理任务中。(2) 以 VideoStorm[7]，Chameleon[10] 等为代表的工作缺乏对网络资源状况的考虑，难以适应动态性较高的物联网应用场景。(3) 以 JCAB[2]，CASVA[1] 等为代表的工作仅仅将场景限制在只有一个边缘服务器中，限制了可用于处理数据流的计算资源。本文在以上工作的基础之上，为了缓解（1）物联网中网络资源稀缺性和动态性的限制（2）不同物联网数据流的不同需求差异性的挑战（3）本地和边缘侧计算资源的综合利用。基于深度强化学习算法实现了一个自适应物联网流处理系统，并且通过将系统进行封装，提供优化目标配置和任务处理配置的接口，使得用户能够仅仅为不同数据流处理任务设定环境，不同的体验质量（QoE）指标以及配置策略集合，即可实现该任务的自适应数据流处理策略优化。本文以物联网边缘计算中的典型应用视频流处理作为测试场景，通过在构建的仿真场景中训练自适应数据流处理策略，在保证低时延和高体验质量的情况下，实现在网络动态变化和计算资源限制下的自适应策略优化。本文的主要贡献点在于：

- 基于深度强化学习算法构建了一个能够应对不同处理需求的自适应流处理系统，通过将系统封装为一个 API，用户仅需设定不同数据流处理任务的配置策略集合，以及优化目标的体验质量（QoE）指标等，即可实现自适应数据流处理策略的优化和迭代。
- 不同于以往的工作将场景限制在仅有一个边缘服务器可以用于数据流卸载处理，本文所设计的系统能够在存在多个边缘服务器的场景中综合利用多个边缘服务器的计算资源进行自适应数据流处理。

1.4 论文结构

为了验证上述研究内容，本文设计并实现了一个基于深度强化学习的自适应物联网数据流处理系统，整篇论文将对此系统的设计与实现进行详细介绍，本文行文组织结构如下：

第一章为引言，主要介绍本工作的研究背景及意义，研究现状概述以及核心的研究内容和贡献点。

第二章为相关研究综述，更详细的梳理和总结了现有的一般流处理系统和框架，部分视频流处理相关的工作，以及与本文实验部分相关的物联网视频流数据处理领域的部分工作，讨论了本文工作与相关工作的不同点和贡献点。然后分析了本文工作的强化学习算法的理论基础。

第三章为基于强化学习的物联网自适应流处理方法，首先，对问题进行了分析和形式化。然后，详细介绍了流处理系统的设计，编程模型的抽象和设计。接着，以物联网数据流中典型的视频流作为例子，形式化问题实例，分析该问题场景中的设计细节。最后，分析了本文流处理系统的深度强化学习算法设计。

第四章为系统实现和性能评估，主要描述了系统各个模块的具体实现，测试并分析在不同网络环境下的实验结果。

最后是结论，参考文献，项目代码和致谢。

2 相关工作综述

在这一章，我们首先分析通用的实时流处理的相关技术，并分析通用的实时流处理技术难以在物联网流处理中应用的原因。然后讨论物联网中的部分基于边缘计算的流处理技术及其优缺点。接着，着重分析物联网中典型的视频流处理的相关工作与技术。

2.1 实时流处理技术

针对广泛的流式数据处理，有许多著名的开源框架和产品。Spark Streaming[5] 通过将数据流划分成微批次，然后将这些微批次送到 Spark 引擎中进行处理，这使得 Spark Streaming 可以实现低延迟和高吞吐量的流处理。Apache Flink[6] 通过支持有状态的计算、事件时间处理和窗口处理等功能。Flink 的流处理是基于数据流的，可以实时处理无限量的数据，支持基于时间和其他条件的数据窗口，提供了比 Spark Streaming 更高级的处理能力，适用于对实时性能有高要求的应用场景。Hazelcast[11] 等开源框架通过提供分布式内存、数据网格、分布式集合等分布式服务来支持分布式数据流处理，可以处理大规模的数据，具有高可用性和高性能。Hazelcast 还提供 Hazelcast Jet 分布式流处理框架，支持基于事件时间和处理时间的窗口计算，可以将数据流分布到多个节点上进行并行处理，实现低延迟和高吞吐量的流处理。这类通用的流数据处理框架虽然对能够对实时数据流进行高效处理，但是缺乏对网络资源的感知，难以适应复杂多变的物联网无线网络环境。此外，这些流处理框架往往运行在计算资源丰富的集群上，物联网设备虽然已经具备一定的计算资源，但是仍然难以承载这类框架的运行。

2.2 物联网实时流处理技术

除了针对广泛场景的流式数据处理框架，还有部分基于云的物联网数据处理系统。在这类系统中，大部分数据都会被发送到云端进行分析。今天，因为云环境可以提供充足的计算资源，许多计算密集型的物联网应用程序 Google Nest Cam[12] 和 Netatmo[13] 采用了这种模型。但是，这样的解决方案不能应用于对时延非常敏感的

物联网流应用，因为（1）网络的波动可能会导致较高的时延并且会对回传信息的网络带宽造成压力（2）将敏感数据卸载到第三方云提供商可能会导致隐私问题。在物联网应用中，部分应用需要在一定的时延限制内进行处理，例如实时视频流监控和实时物流追踪等。如果将数据全部发送到云端进行处理，这些应用将无法满足要求。因此，需要开发一种新的解决方案，能够处理时效性要求高的物联网流应用，同时又能保证数据隐私和安全。在此基础之上新的解决方案是在边缘设备上进行数据处理和分析。边缘设备通常位于物联网边缘，可以直接收集和处理数据，而不需要将所有数据发送到云端。这种模型具有许多优势，包括降低网络带宽的使用、降低延迟、提高数据安全性等。同时，边缘设备还可以利用本地存储和计算资源，避免过度依赖云计算资源，降低成本。因此，边缘计算已经成为解决物联网流应用的重要手段。针对物联网边缘计算场景，在工业界，亚马逊，阿里巴巴，腾讯，微软，谷歌等头部云服务提供商都在物联网领域投入了大量的研究资金和研究人员，也有大量服务和应用落地。阿里云推出了物联网服务平台，通过统一平台提供的设备管理服务集中解决物联网设备的生命周期，设备注册，认证，绑定，授权等问题。该平台还可以处理从物联网中收集的数据，支持实时数据处理，并为开发者提供了远程调试和在线监控等功能，为车联网，智能农业等场景提供新的解决方案。亚马逊也推出个功能强大、可靠性高、安全性好的物联网服务平台（AWS IoT），为开发者和企业提供可扩展、安全的物联网解决方案。该平台可以管理和连接数百万个设备，支持设备注册、认证、通信、控制和监测等功能，同时提供多种开发工具和 SDK，包括 AWS IoT Device SDK、AWS Greengrass[14] 等。亚马逊物联网服务还支持多种通信协议和云端服务，以满足不同场景下的需求。同时，该平台还提供多种安全保障机制，包括设备认证、数据加密、访问控制等，以确保设备和数据的安全性。此外，亚马逊还借助高级 AI 和机器学习整合构建了智能 IoT 解决方案。谷歌推出了 Google Cloud IoT Core 全托管的物联网服务，旨在帮助企业轻松连接、管理和监控大规模的物联网设备。该平台与谷歌云的其他服务集成，如 Google Cloud Pub/Sub、BigQuery 等，可以帮助用户进行数据的实时分析和处理，为企业快速构建物联网应用提供支持。

在研究领域，Kay 和 Patrick 等人的工作 Sparrow[15] 设计了一个分布式低延迟的任务调度策略。这些工作在边缘计算的模型基础上，将数据处理放在边缘进行，不需要连接到云后端。通过在边缘侧安装一个中心设备，从其他物联网设备收集数据并进行数据处理。然而，这些解决方案受到中心服务的计算能力限制，无法支持跨

多个设备进行分布式数据并行处理，因此吞吐量有限。一旦中心设备故障，也可能引入单点故障。

2.3 物联网视频流相关工作

针对物联网中的典型应用–视频流数据处理，已经有大量的研究工作。

2.3.1 基于最优化的方法

VideoStorm[7]的场景是大型集群中的视频处理，基于在离线阶段通过生成一个查询资源质量的分析器（query resourcequality profile）来获得所查询资源的质量概况，并为每一个数据资源生成一个 query，在线阶段通过模型预测控制（MPC）为每个 query 分配集群中的处理资源以优化资源质量和多维配置的平衡，权衡质量和查询之间的滞后（lag）。但是 VideoStorm 并未考虑网络带宽对数据上传至数据中心这个过程的影响，忽视了网络带宽变化产生的延迟。Chameleon[10]则是发现视频内容不同对计算资源的需求也不相同，从而提出通过对视频内容的分析来生成视频配置，同时考虑到不同质量的视频对分析模型的需求也不同，为了节约计算资源，文中提出了在视频分析阶段也可以有不同的方案选择，并根据潜在的时间和空间相关性，随着时间的推移和跨多个视频分摊分析成本。该方案同样没有考虑网络带宽对数据传输的影响，更多关注的是处理时计算资源的调度问题。

随着研究推进，研究[16-18]发现网络带宽的稀缺性和动态性问题是视频传输分析系统中的关键问题，网络带宽变化的可能会使得优化配置给用户感知带来级联式的影响，降低用户的感知体验。此外，无线蜂窝网络等所能提供的带宽也有一定限制。虽然近年来 5G 技术逐渐普及，但是距离 5G 技术大规模成熟地应用还有一定差距。同时接入设备增多，包括直播，短视频等新的应用也给视频传输实时性提出了更高的要求。一些工作开始关注到不仅仅是单客户端单服务器的场景。Awstream[4]考虑广域网场景中网络带宽变化的影响，提出了一个易用的能够同时实现低延迟和高准确性的推理模型。模型通过结合离线和在线两个分析阶段，自动学习一个平衡准确性和带宽的策略。然后在运行时根据实时的网络带宽调整应用的数据率，同时尽可能保证服务器上视频分析任务的准确度。JCAB[2]在多客户端单边缘服务器的场景中，基于李雅普诺夫优化和马尔可夫近似，提出 JCAB 在线算法，通过将算法部署在服务器端进行联合优化配置和带宽分配，控制每一个客户端上的卸载策略。部分解决

基于边缘的视频分析系统中的边缘容量限制、未知网络变化、视频内容动态等问题，实现了在保证低延迟的前提下，平衡了视频流分析的准确性和能耗。

2.3.2 基于深度强化学习的方法

近年来基于深度强化学习的方法被成功应用到物联网和边缘计算领域，在物联网中的序列决策问题中展现出巨大潜力。不同于最优化方法，深度强化学习可以无需对环境进行复杂建模，仅仅用对环境的部分观测来描述当前环境的状态，根据对环境的观测推理得到每个配置，隐式地学习环境和对应配置的内在联系和影响。通过设定不同的优化目标来实现对决策的优化。此外，深度强化学习不同于基于最优化方法分为离线在线两个阶段，能够以端到端的方式完成推理，在保证推理结果的准确性和实时性的同时能够更简单地应用到各种不同的任务中。

2017 年，Pensieve[3] 首次将深度强化学习方法引入到 DASH 协议[9] 的 ABR 算法中，Pensieve 构建了一个基于深度强化学习 A2C 算法的 ABR 算法并替换了 DASH 协议中基于规则的方法，根据对网络带宽，缓存等的观测选择不同的码率配置，实现了在网络带宽的限制下用户体验感知（QoE）的最大化。在 Pensieve 的基础上，Comyco[19] 通过模仿学习（imitation learning），依靠专家经验的先验来提升了深度强化学习的环境采样效率。Rldish[20] 发现在某些场景下每个视频的第一块能够一定程度上反应整个视频提供的 QoE，提出通过深度强化学习算法实现通过对视频流第一块的分析来为新在线视频用户确定视频流配置。iView[21] 在 360° 可交互视频场景下，结合多模态在环境感知阶段获得了提取特征后的信息。除了优化用户的服务体验，物联网领域中还有一部分任务是在边缘服务器中进行任务分析，相同的视频配置对于用户观看体验和视频分析模型有不同的影响。CASVA[1] 在单服务器单客户端场景下，将服务器端的视频分析模型推理的准确性作为新的优化目标，以此来优化对视频配置传输的决策。

2.3.3 现有工作的不足

在深度强化学习之前，大多数研究基于各类经典优化算法。通过对环境的合理建模，构建环境和优化目标（包括用户 QoE，时延等）的联系，将推理过程分为离线和在线两个阶段分别进行处理。虽在早期的场景下取得了较好的效果，但是该类方法先现今的物联网流处理场景下表现出一些难以克服的不足。

- 大规模接入数据源和网络状况的波动性和稀缺性使得对环境建模变得极其困难，建立在不完善环境模型上的经典优化算法往往只能得到相对次优的策略。
- 随着经济科技发展，用户对体验的要求提高，而各类应用提供的服务不同，传统优化算法针对不同的问题需要重新建模分析，这使得基于优化的方法难以应对不同应用的不同需求

深度强化学习无需对环境复杂环境建模和先验且通过神经网络隐式学习环境的模式更具鲁棒性，在此之上只需要构建不同的优化目标即可应用于不同的领域。

2.4 深度强化学习技术概述

强化学习方法已经在时序决策问题上取得了巨大进展，不同于机器学习中的监督学习和无监督学习，强化学习是通过于环境不断交互，来不断优化自身策略的方法。强化学习问题是建立在环境可以抽象为一个马尔可夫决策过程（Markov Decision Process，MDP）之上的。一个马尔可夫决策过程可以用一个五元组来表征 $< S, A, R, P, \rho_0 >$。

- S 是所有状态（state）的集合
- A 是所有动作（action）的集合
- R: $S \times A \to \mathbb{R}$ 是奖励函数，t 时刻的奖励函数 r_t 由 s_t a_t 决定
- P: $S \times A \times S \to \mathbb{R}$ 是状态转移函数，给定当前状态 s_t，采取动作 a_t 后，转移到新状态 s_{t+1} 的概率。
- ρ_0 是初始状态的概率分布，$\sum_{s \in S} \rho_0(s) = 1$

如图2.1所示，强化学习的智能体与环境基于离散的时间步作用。在每一个时间 t，智能体获得一个观测 o_t，通常其中包含奖励 r_t。然后从允许的动作集合中选择一个动作 a_t，在环境中执行。环境观测到一个新的状态 s_t，然后决定了和这个变化 (s_t, a_t, s_{t+1}) 相关联的 r_{t+1}。强化学习主体的目标，是得到尽可能多的奖励。主体选择的动作是其历史的函数，它也可以选择随机的动作。

在本文所设定的场景及其他不完全信息对弈的场景中，智能体无法观测到整个环境的状态，该过程被定义为部分可观测马尔可夫决策过程（Partially Observable Markov Decision Process，POMDP）。在整个交互过程中，智能体智能观测到部分状态 o_t，为环境状态 s_t 的一部分。

通过定义整个交互过程的累计回报 G_t，我们得到强化学习算法的优化目标，即

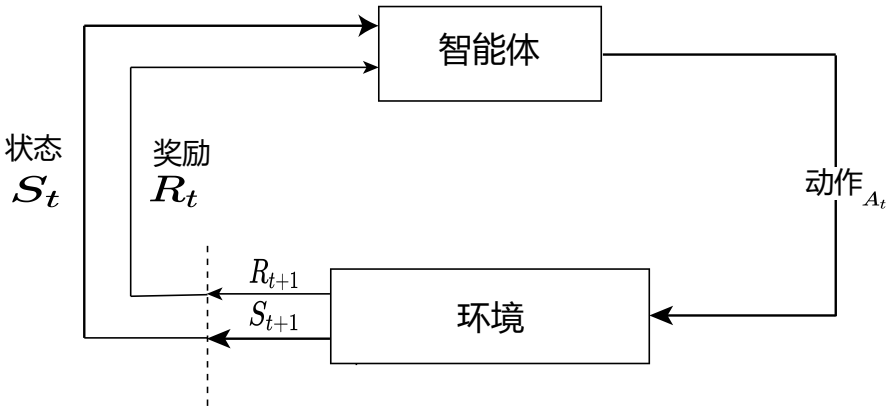

图 2.1　强化学习智能体与环境交互过程

最大化每一个时间步的累计回报的期望值。G_t 和优化目标 θ 形式化如下。

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i \tag{2.1}$$

$$\theta^* = \arg\max \mathbb{E}_{\pi_\theta}[G_T] \tag{2.2}$$

(2.1) 中 γ 表示折扣因子，是智能体对长短期回报的权重分配。(2.2) 中 π_θ 是以 θ 为参数的策略 π，在深度强化学习中，θ 往往表示表征策略的神经网络中的参数。深度强化学习通过梯度上升方法，不断调整策略来最大化累计回报。

深度强化学习算法大致可以分为三个类型：基于价值的方法，基于策略的方法，以及结合价值和策略的方法。

基于价值的深度强化学习是一种经典的强化学习算法，其核心思想是通过学习价值函数来指导智能体做出最优决策。价值函数可以有两种形式，一是状态价值 $V(s_t)$ 该式表示在状态 s_t 是所能获得的长远奖励期望，二是状态动作价值 $Q(s_t, a_t)$，该式表示在状态 s_t 采取动作 a_t 所能获得的价值。

2.4.1　基于价值的强化学习方法

基于价值的深度强化学习算法的核心思想是使用神经网络来拟合价值函数 $Q^*(s, a; \theta)$，然后根据拟合的价值函数来指导智能体做出最优决策。以 DQN[22] 为例，基于价值的深度强化学习算法的训练过程可以分为以下几个步骤：

1. 初始化神经网络参数 θ。

2. 在每个时间步 t，根据当前状态 s_t，使用神经网络预测每个动作的价值 $Q(s_t, a_t; \theta)$。
3. 根据 ϵ-贪心策略选择动作 a_t。
4. 执行动作 a_t，获得奖励 r_t 和下一个状态 s_{t+1}。
5. 根据下一个状态 s_{t+1}，使用神经网络预测每个动作的价值 $Q(s_{t+1}, a_{t+1}; \theta)$。
6. 根据下一个状态的最大价值，计算目标值 $y_t = r_t + \gamma \max_a Q(s_{t+1}, a; \theta)$。
7. 使用均方误差作为损失函数，即 $\mathcal{L}(\theta) = \frac{1}{2}(y_t - Q(s_t, a_t; \theta))^2$。
8. 更新神经网络参数 θ，即 $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$，其中 α 是学习率。

通过以上训练过程，神经网络可以逐渐拟合出价值函数 $Q^*(s, a; \theta)$，从而指导智能体做出最优决策。需要注意的是，在训练过程中，为了平衡探索和利用，通常会采用 ϵ-贪心策略。

2.4.2 基于策略的强化学习方法

基于策略的深度强化学习（Policy-based Deep Reinforcement Learning）是另一种常见的深度强化学习算法。和基于价值的方法不同，基于策略方法不直接预测每个状态的价值函数，而是直接学习策略函数。策略函数直接映射状态到动作的概率分布，即 $\pi(a|s)$，表示在状态 s 下，选择动作 a 的概率。

基于策略方法的优点是可以直接优化最终要执行的动作，因此不需要计算状态值函数或动作值函数。相比于基于价值的方法，基于策略的方法可以更好地处理连续动作空间和高维状态空间。

基于策略方法的通过定义一个损失函数，用于衡量策略函数的性能。一个常见的损失函数是策略梯度（Policy Gradient），表示为：

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log \pi_\theta(s, a) Q_{\pi_\theta}(s, a)] \tag{2.3}$$

其中，$J(\theta)$ 表示策略函数的总体性能，θ 表示策略函数的参数，$\pi_\theta(s, a)$ 表示在参数为 θ 的策略函数下，在状态 s 下选择动作 a 的概率，$Q_{\pi_\theta}(s, a)$ 表示在参数为 θ 的策略函数下，在状态 s 选择动作 a 的期望回报。策略梯度的优化可以使用随机梯度上升（Stochastic Gradient Ascent）或其变体。通常，可以使用神经网络来实现策略函数。神经网络的输出层可以是一个概率分布，例如 softmax 函数。

以 REINFORCE[23] 算法为例，基于策略的深度强化学习算法通常包括以下迭代

步骤：

1. 初始化策略：随机初始化策略函数 $\pi_\theta(a_t|s_t)$ 的参数 θ。
2. 数据采集：使用当前策略 π_θ 与环境进行交互，收集经验样本 $\{s_t, a_t, r_t, s_{t+1}\}$。
3. 计算损失：根据策略梯度定理，利用采集的样本计算损失函数 $J(\theta)$，通常采用策略梯度方法，其中损失函数的形式为：$J(\theta) = \mathbb{E}_{s_t\sim\rho^\pi, a_t\sim\pi_\theta}[log\pi_\theta(a_t|s_t)Q^\pi(s_t, a_t)]$，其中 ρ^π 是当前策略下的状态分布，$Q^\pi(s_t, a_t)$ 是状态-动作对 (s_t, a_t) 的长期期望回报，可以通过蒙特卡罗方法或者基于值函数的方法进行估计。
4. 更新策略：根据计算得到的梯度信息对策略参数 θ 进行更新，通常采用随机梯度下降（SGD）或者自适应优化方法，如 Adam、RMSprop 等。
5. 重复步骤 2-4，直到策略收敛或者达到最大迭代次数。在迭代的过程中，可以采用一些技巧来提高算法的性能，如基于经验回放的方法、多步骤更新等。

2.4.3 演员评论家结构的算法

演员评论家算法是一种结合了策略梯度和价值函数估计的深度强化学习算法。它通过同时学习一个策略网络和一个价值函数网络，来提高强化学习智能体的性能。演员评论家算法的基本思想是，将一个强化学习问题转化为两个问题：一个是评估当前状态的价值（评论家），另一个是在当前状态下如何选取合适的动作（演员）。其中，演员负责策略的生成，评论家负责对策略进行评估和改进。演员网络接收状态作为输入，并输出一个概率分布，用于选择下一个动作。演员网络的参数会根据策略梯度的方法进行更新。评论家网络接收状态作为输入，并输出一个值函数，用于评估该状态的价值。评论家网络的参数会根据 TD 误差的方法进行更新。演员评论家算法通常使用 TD 误差来更新评论家网络的参数，使用策略梯度来更新演员网络的参数。演员和评论家网络都可以采用深度神经网络来实现。结合值函数和策略函数的优点，演员评论家算法可以有效地解决强化学习中的高方差和高偏差问题。

以演员评论家作为框架的算法通常包括以下几个步骤：

1. 定义状态 s 下采取动作 a 的动作值函数为 $Q(s, a)$，以及状态 s 下采取动作 a 的策略为 $\pi(a|s)$。
2. 定义演员神经网络的参数为 θ，输出动作 a_t 的概率为 $\pi_\theta(a_t|s_t)$。
3. 定义评论家神经网络的参数为 ω，估计状态值函数 $V^\pi(s_t)$。
4. 定义时序差分（Temporal Difference, TD）误差为：$\delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$。

其中，r_t 为时刻 t 的奖励，γ 为折扣系数。

5. 更新评论家神经网络的参数，使其能够更准确地估计值函数：

$$\Delta\omega = \alpha_\omega \delta_t \nabla_\omega V^\pi(s_t)$$

其中，α_ω 为评论家神经网络的学习率。

6. 更新演员神经网络的参数，使其输出更好的动作概率分布：

$$\Delta\theta = \alpha_\theta \delta_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

其中，α_θ 为演员神经网络的学习率。

7. 更新演员神经网络的参数后，再使用新的参数更新策略，即 $\pi_\theta(a_t|s_t)$。
8. 重复步骤 4-7 直到训练结束。

在本文所设定的场景及其他不完全信息对弈的场景中，智能体无法观测到整个环境的状态，该过程被定义为部分可观测马尔可夫决策过程（Partially Observable Markov Decision Process，POMDP）。在整个交互过程中，智能体智能观测到部分状态 o_t，为环境状态 s_t 的一部分。如2.4所述，深度强化学习算法大致可以分为以下三种，基于策略的方法（policy-based），如 REINFORCE[23]；基于价值的方法（value-based），如 DQN[22]；演员评论家方法（actor-critic），如 A3C[24]，TRPO[25]，PPO[26] 等。基于价值的方法通过对环境的状态价值或者状态动作价值进行评估，智能体只需要根据价值函数选择当前能够使得当前状态价值最高的动作即可。基于策略的方法则是根据对当前环境状态直接输出动作的概率分布，直接使用奖励进行梯度上升优化网络。基于价值的方法通常和贪婪策略一起使用，当最优策略是随机策略或者动作空间维度较高时，可能表现不佳。基于策略的方法虽然直接输出动作的概率，能够有效应对高维动作空间，但是梯度上升的方法也可能使其陷入局部最优。为了平衡这两种方法的优缺点，演员评论家的方法结合了两种算法的思想。相比以价值函数为中心的算法，演员评论家算法应用了策略梯度的做法，这能让它在连续动作或者高维动作空间中选取合适的动作。相比单纯策略梯度，演员评论家算法应用了其他策略评估的做法，使其能进行单步更新而不是回合更新，比单纯的策略梯度算法的效率要高。基于以上原因，本文采用基于演员评论家框架的算法。以 A2C/A3C[24] 为例，演员（actor）与环境交互，并在评论家（critic）的价值函数的指导下用策略梯度学习一个更好的策略。评论家（critic）要做的是通过演员（actor）与环境交互收集

的数据学习一个价值函数用于评估在当前状态下动作的好坏，进而帮助演员进行策略更新。具体地，本文最终采用 PPO[26] 算法，PPO 是应用广泛同时性能稳定，已经应用在了许多现实场景中，包括最新的 ChatGpt 的训练中也使用了 PPO 算法。本文将在后文详细介绍系统的 PPO 算法设计。

3 基于深度强化学习的物联网自适应流处理方法

这一部分，我们首先以物联网中的典型视频流处理应用为例，通过分析部分预实验结果，论述本文的研究动机。然后详细介绍流处理系统的架构和编程模型。最后以物联网中典型的视频流卸载问题作为场景，将问题和系统形式化。

3.1 问题分析

近年来，部分工作提出了自适应流媒体来应对网络条件的变化。其核心思想是根据网络状况调整传输数据量，同时保持令人满意的应用程序性能。这是基于一个观察结果，即流媒体数据的不是每个维度都对应用程序的最终结果有显著的贡献。

以物联网中典型的视频流处理任务为例，对于行人检测应用程序，分辨率对检测精度的影响远高于帧率。当带宽不足时，可以适当降低帧率来减少数据量而不牺牲性能。现有的自适应流媒体方法[27]允许用户手动开发此类降级策略，以确保在较差的网络条件下进行实时分析。例如，“默认情况下，从 CCTV 摄像头发送最高保真度的所有图像到数据中心。如果带宽不足，则切换到以 75% 的保真度发送图像，然后在仍然没有足够带宽的情况下，以 50% 的保真度发送图像。超过这一点，降低帧率，但保持图像的 50% 保真度”。然而，这样的手动策略在面对动态网络条件时过于粗粒度和缺乏灵活性。最先进的方法 AWStream[4]为用户提供 API，以灵活地开发一组可能的降级策略，并通过穷尽搜索选择最佳策略。然而，AWStream 的性能依赖于预定义的、有限的和离散的策略空间，使其对于网络条件快速变化和波动较大的物联网场景不够灵活。粗粒度的手动策略缺乏对计算资源和网络带宽的综合利用，而预定义的、有限的策略空间又难以适应网络波动较大的无线网络场景。此外，在视频流处理中，视频流的配置主要包含三个关键的参数设置：分辨率（resolution），帧率（framerate）和视频编码的量化指标（quantizer）。一方面，这三个参数的常用配置就可以组合出上百种配置，在加上卸载目标这一决策维度，所能够产生的配置高达数百种，基于预定义的、有限的决策空间的方法难以适用。另一方面，通过调节这三个参数，我们可以控制视频流的质量和大小，但是由于 H.264 等视频编码协议的特点，不同的编码配置对视频流编码的影响不一定是线性的。我们将视频编码的

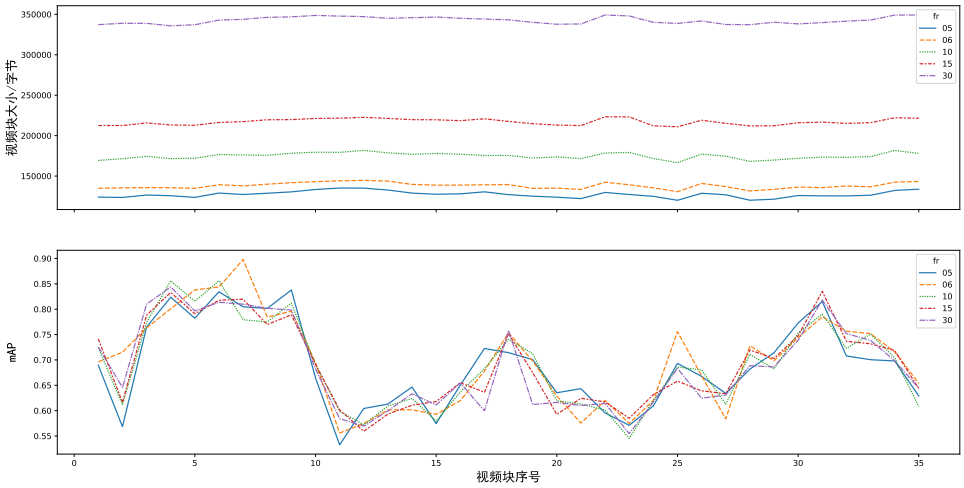

图 3.1　编码后视频块大小与编码配置的关系

分辨率和量化指标分别固定为 960p 以及 23，观察不同的帧率对视频块大小和 mAP 的影响，我们以 MOT16-04[28] 的数据集作为视频帧，将其划分为每 30 帧一个视频块，一共 35 个视频块，我们使用 gstreamer 以及其 x264enc 插件对这 30 个视频块所对应的帧，在固定的 960p 分辨率和 23 的量化指标下分别进行了不同帧率的视频编码，实验结果如图3.1所示。从实验结果可以看出，（1）在保持分辨率和量化指标相同的情况下，降低配置的视频的帧数，不同帧率的视频块的大小都出现明显的下降趋势，整体表现出以量化标准每降低 50%，视频块大小减少大致 30%-50% 的规律；（2）在保持分辨率和量化指标相同的情况下，帧率的高低对视频块的目标检测结果的影响并不是线性的。帧率高并不意味着目标检测的精度高。这也加大了寻找最优化策略的难度。

在大规模设备接入的物联网实时数据流卸载的场景中，大量接入设备之间的异构性，数据流种类的多样性以及网络资源的稀缺性和动态性都使得我们难以对环境的状态转移进行准确建模。为了部分缓解环境难以建模的挑战，部分工作的模型基于网络带宽可实时预测这一假设，而这一点难以在真实的网络环境中实现。其次，物联网实时数据流的卸载任务在时序上是相关的，可以抽象为一个部分可观测马尔可夫决策过程（partially observable Markov decision process，POMDP）。

基于以上观察和挑战，本文采取基于深度强化学习的方法，通过对每次数据流卸载过程中环境状态（包括传输一段数据流的平均网络带宽，网络带宽波动方差，数据流处理时延等）的观测作为环境的表征，构建一个仅包含部分环境观测的仿真系

统，基于无模型的深度强化学习算法实现对数据流卸载策略的优化。针对不同的处理需求和优化目标，用户可以通过使用系统封装的 API，设定系统的环境观测空间、决策空间以及决策奖励设计，从而快速地实现不同需求的自适应流处理。在后文中，我们以物联网中典型的视频流目标检测任务作为场景，介绍系统的设计和实现。

3.2 基于深度强化学习的物联网流处理系统的架构

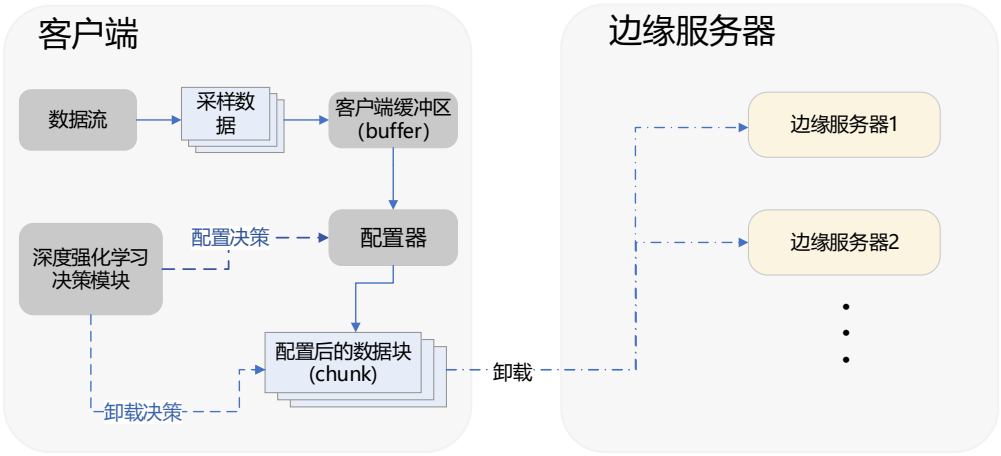

图 3.2　基于深度强化学习的物联网流处理系统架构

图3.2描述了基于深度强化学习的物联网数据流处理系统框架。系统分为三个主要模块客户端模块，边缘服务器模块，以及网络仿真模块。在本文描述的物联网实时数据流处理系统中，每一个单位处理时间内，客户端设备会产生或者从外界实时采集数据，并缓存到设备缓冲区中。深度强化学习算法模块会根据当前的客户端设备的内部信息以及对客户端所处环境的网络的感知信息生成一个配置决策，按照配置决策从缓冲区获取一定的数据并进行相应的配置处理。然后，系统按照配置决策给出卸载目标，选择将数据块卸载到合适的边缘服务器中进行后续的任务处理，以此实现降低时延的同时尽可能提高分析的准确率。

3.3 编程模型

由于现有的物联网流处理系统大多仅仅支持特定的任务，而使用这些任务下训练的模型和代码框架难以快速应用到其他任务和优化目标中去，为物联网流处理系统的普适性和鲁棒性产生很大的阻碍。RL-adapt[30] 首次为物联网流处理系统设计了

表 3.1　物联网流处理系统编程模型

API	描述
set_config_set	*set_config_set*() 接口用于用户自定义可选配置的集合，通过用户自己设定的可选配置集合，系统能够自由组合出所有可能的配置并从中选择。
set_env	*set_env*() 接口用于用户自定环境的观测空间，通过简单配置环境以及环境中的可观测状态空间，将环境的状态转移问题转化为一个马尔可夫过程或者部分可观测的马尔可夫过程。
set_evalution	*set_evaluation*() 接口用于用户自定义可选的体验质量（QoE）指标，通过定义影响用户体验质量的指标，并自定义不同指标的权重，系统可以根据指标对算法进行优化。
set_algorithm	*set_algorithm*() 接口用于用户自定义强化学习算法，现阶段支持深度强化学习框架天授[29]中所支持的算法。

统一接口的编程模型，使得该框架能够广泛适用于各类型的物联网流处理系统的应用中。但是该框架对存在多个边缘服务器的场景或者本地设备也能作为卸载目标的场景所提供的支持并不完善，并且提供的 API 接口也并不直观，不方便开发人员使用和修改。因此基于该工作的思路，本系统在深度强化学习库 tianshou[29] 的基础上，提供了一个新的物联网流处理系统编程模型，该编程模型的接口与深度强化学习算法中的各组成部分和物联网流处理系统中的一些关键状态（例如，带宽）对应，用户仅需要按照 OpenAI gym 的深度强化学习环境设计标准即可快速适配并应用到各个物联网流处理系统场景中。核心 API 的设计和功能如表3.1所示，使用示例如图3.3所示。

3.4　问题形式化

本小节中，我们将流处理系统任务设定为物联网中典型的视频流卸载和分析场景。以物联网中视频流处理任务为例，详细分析和讨论如何在上文的编程模型基础上开发并测试物联网流处理系统。系统的设计架构如图3.4所示。

```python
class SimEnv(gymnasium.Env):
    # user-defined function
    def step(self, action):
        pass
    def reset(self, seed):
        pass
    def terminated(self):
        pass
    def truncated(self):
        pass

def qoe_metric():
    pass

pipeline = AdaptionPipeline()
pipeline.set_config_set(spaces.Discrete(config_num))
pipeline.set_env(SimEnv)
pipeline.set_evaluation(qoe_metric)
pipeline.set_algorithm("ppo")
pipeline.run()
```

图 3.3　系统使用示例

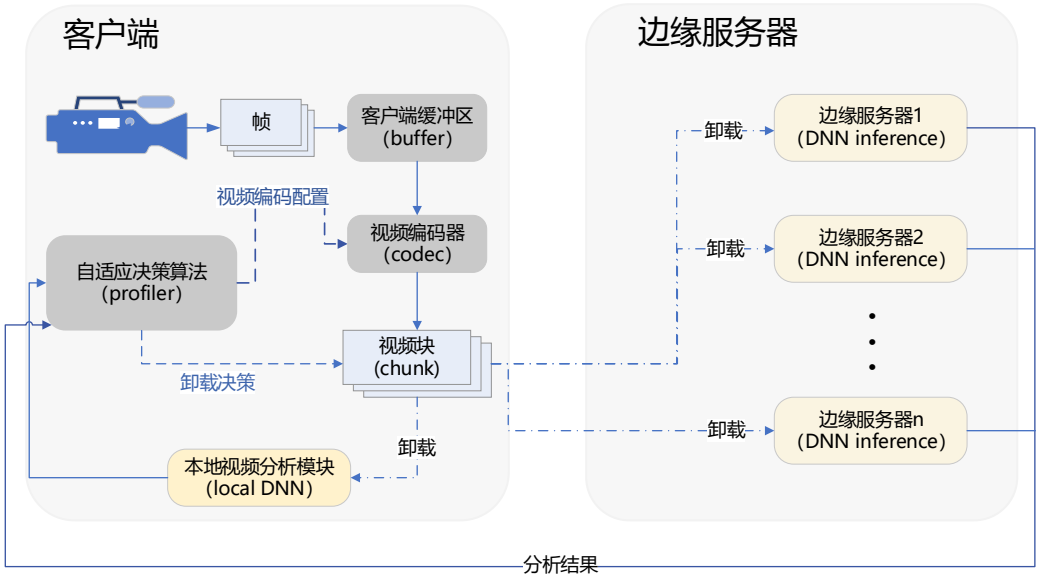

图 3.4　系统架构（以典型的物联网视频流应用为例）

如图3.4所示，假设我们现在有一个实时视频流，我们的客户端设备（智能摄像头）可以实时捕获视频帧数据并将其编码封装传输给其他设备。客户端有 K 条网络链路可以连接到最近的 K 个不同的边缘服务器，再加上客户端设备本身具备一定的视频流处理分析能力，即我们有 $K+1$ 个可供卸载的目标 $\mathcal{U}=\{u_1,u_2,...,u_{K+1}\}$。不同的卸载目标的可利用的计算能力不同，所以在其上部署的视频流分析模型的分析处理能力也不同。我们假设客户端仅具备初步的处理视频分析任务的能力，因此客户端本地只能部署一个参数较少的模型。相较于客户端本地，边缘服务器中计算资源较为充足，能够支持更大的分析模型。但是不同的边缘服务器的计算资源和任务负载也有差异，为了简化环境的复杂度，我们在不同的边缘服务器总部署了不同的参数的模型。部署的所有模型集合可表示为 $\mathcal{M}=\{m_1,m_2,...,m_{K+1}\}$。对于环境状态转移过程，我们将环境的单位步长（step）定义为卸载一个视频块的过程，即从缓存中的一个视频块并发送到卸载目标的过程。在每一步中，客户端从缓存中取出一个视频块，自适应决策算法根据视频块的信息和当前环境的观测，生成视频块编码配置和卸载目标，然后将编码后的视频块发送到卸载目标。在服务器端（卸载目标），接收到客户端传输的视频块后，即对视频块进行分析，并将分析的结果回传给客户端，分析结果是客户端评估所采取策略的好坏的重要指标之一。通过自定义优化目标，即为时延和任务分析准确率设置不同的权重，系统在不断为新的视频块生成给配置并卸载到边缘端分析的过程中，不断迭代优化策略，从而实现在用户目标定义下的低时延，任务高精度和低能耗三者的权衡。

图3.5描述了系统生成视频配置的演员评论家算法的框架。深度强化学习中使用一组向量来描述环境当前状态。在对网络环境的仿真中，在本文的场景下由于我们无法模拟现实网络环境的所有状态，只能通过模拟对网络环境的部分观测将环境状态的转移构建为一个部分可观测的马尔可夫决策过程（POMDP），在每一个单位步长中发送一个视频块。模拟环境的状态 s_t 定义为：

$$s_t=(m_t,v_t,b_t,s_t,d_t) \tag{3.1}$$

其中，m_t 表示在发送上一个视频块的过程中的平均网络带宽；v_t 表示在发送上一个视频块的过程中的网络带宽的方差，从一定程度上反应网络带宽的波动情况；b_t 表示当前客户端中的缓存剩余空间；s_t 表示上一个视频块的大小；d_t 表示发送的上一个视频块经过传输分析处理所消耗的总时延，总时延包括（1）对捕获的视频帧编

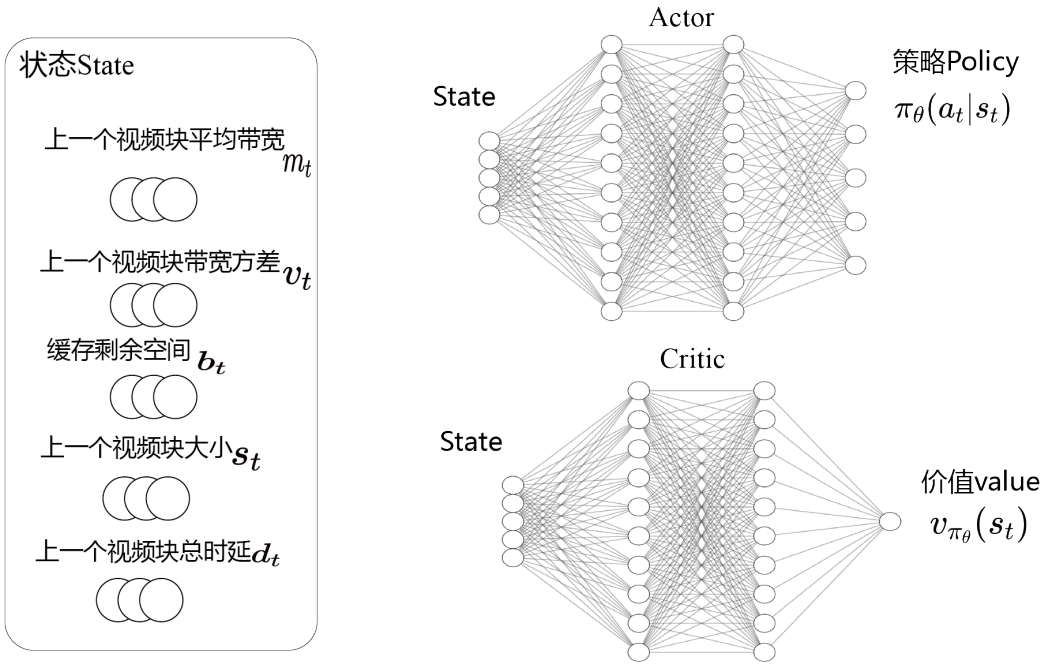

图 3.5　系统使用的生成配置的 PPO 算法

码封装成视频块的时延（2）在网络中传输视频块的时延（3）视频块分析的时延。

对于每一个环境观测 s_t，客户端上运行的 PPO 算法决定采取哪个动作。动作空间 $a_t = (framerate, resolution, quantizer, target)$ 包括四个参数：（1）帧率（2）分辨率（3）量化指标（quantizer）（4）卸载目标，即将该视频块卸载到哪个边缘边缘服务器。常见的视频流传输协议中，帧率，分辨率，量化指标往往作为取一些固定的组合（knob）作为配置集合，但是由于帧率，分辨率和量化指标对深度神经网络的分析影响不同，我们将其作为独立的配置进行决策。在物联网场景下的视频流处理任务有三个重要的指标：（1）时延、（2）分析精度、（3）能耗。系统的奖励函数综合了三个指标的影响，并通过为每个指标赋予不同的权重来表征优化目标的差异性。奖励函数的设计如下。

$$QoE = mAps \times framerate \times \alpha - delay \times \beta - energy/norm \tag{3.2}$$

3.5　问题求解

在获得对环境的观测 s_t 之后，客户端需要根据当前状态和策略并选择一个动作分布并根据探索和利用的原则选择一个工作 a_t。策略是一个根据当前状态生成动作概率分布的神经网络，表示为 $\pi_\theta = p(a_t|s_t) \to p \in [0, 1]$。

由于在仿真环境中定义的观测状态空间非常大，无法完全记录，所以在优势演员评论家算法中策略和评估都是由神经网络表征的，我们将这个表征策略的网络称为演员网络（Actor），演员负责与仿真环境交互，在客户端发送第一个视频块之后获得初始状态，将状态 s_t 输入到演员网络（Actor），根据当前环境的状态做出相应的卸载动作概率分布。即输入当前对环境的观测状态 s_t，输出一个动作的概率分布 $p(a_t|s_t)$，每个概率对应每一个卸载动作，该过程将观测映射为动作概率分布即卸载策略 $\pi_{\theta_1}(s_t)$，θ 是神经网络中的参数。从当前演员网络表征的卸载策略输出的卸载动作概率分布中采样一个确定的动作之后，系统按照卸载动作将视频块进行编码配置并传输到相应的边缘服务器上。在获得了边缘服务器的分析结果之后，需要对当前卸载决策动作的优劣进行评价，为此演员评论家算法引入了一个对观测状态进行评估的评论家网络（Critic），评论家网络通过输入环境的观测 s_t，生成对于环境状态价值的评估，即当前状态在未来所能获得的奖励的期望，表示为 $V_{\theta_2}(s_t)$。

每当仿真环境转移一步，系统得到一个环境状态 s_t，按照策略 π_{θ_2} 生成的卸载动作动作分布选择一个卸载动作 a_t，并且采取这个卸载动作之后得到一个即时奖励 r_t，根据时序差分的方法我们分别对演员和评论家进行更新。演员网络的更新方法如下：

$$\nabla \bar{R}_{\theta_1} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(r_t^n + v^{\pi_{\theta_2}}(s_{t+1}^n) - v^{\pi_{\theta_2}}(s_t^n))\nabla log p_{\theta_1}(a_t^n|s_t^n) \tag{3.3}$$

$$\theta_1 \leftarrow \theta_1 + \alpha\nabla \bar{R}_{\theta_1} \tag{3.4}$$

其中，θ_1 表示演员网络的参数，N 表示采样的序列个数，T_n 表示每个采样序列的总时间步，θ_2 表示评论家网络的参数，α 表示学习率，$r_t^n + v^{\pi_{\theta_2}}(s_{t+1}^n) - v^{\pi_{\theta_2}}(s_t^n)$ 叫做优势函数（advantage function），该函数表征一个确定的动作比当前策略下所有“动作平均”好多少。

评论家网络的更新方法如下：

$$\mathcal{L}(\theta_2) = \frac{1}{2}(r_t^n + \gamma v^{\pi_{\theta_2}}(s_{t+1}) - v^{\pi_{\theta_2}}(s_t))^2 \tag{3.5}$$

$$\nabla\mathcal{L}(\theta_2) = -(r_t^n + \gamma v^{\pi_{\theta_2}}(s_{t+1}) - v^{\pi_{\theta_2}}(s_t))\nabla v^{\pi_{\theta_2}}(s_t) \tag{3.6}$$

$$\theta_2 \leftarrow \theta_2 + \alpha\nabla\mathcal{L}(\theta_2) \tag{3.7}$$

其中，θ_2 表示评论家网络的参数，$\mathcal{L}$ 表示评论家网络的损失函数，$\gamma \in [0,1]$ 表

示折扣因子，反应未来奖励的重要程度；α 表示学习率。

Algorithm 1 基于 A2C 算法的系统策略优化过程

初始化演员和评论家网络的参数 θ_1 和 θ_2

对于每一个采样的序列：

在发送缓存中的第一个视频块之后，系统获得环境的初始状态 $s_t = (m_t, v_t, b_t, s_t, d_t)$

当 s_t 不是序列终点时执行：

获得动作概率分布 $A \sim \pi_{\theta_1}(\cdot|s_t)$

从动作概率分布中采样得到动作 a_t，得到新的环境状态 s_{t+1} 和即时奖励 r_t

计算优势函数 $A^{\pi_{\theta_1}}(a_t|s_t) = r_t^n + v^{\pi_{\theta_2}}(s_{t+1}^n) - v^{\pi_{\theta_2}}(s_t^n)$

更新评论家网络，$\theta_2 \leftarrow \theta_2 + \alpha\nabla\mathcal{L}(\theta_2)$

更新演员网络 $\theta_1 \leftarrow \theta_1 + \alpha\nabla\bar{R}_{\theta_1}$

更新状态 $S \leftarrow S'$

在实验中，我们发现 A2C 中的策略梯度方法可能会导致过大的策略更新和较低的采样效率，因此我们考虑使用近端优化剪裁的策略（PPO clip）[26]。近端策略优化在结合 AC 框架在 TRPO[25] 基础之上，通过重要性采样且直接将 KL 散度纳入计算目标而非约束，维持了目标策略和示范策略输出的分布相差在较小区间的同时，相比 TRPO 减少了计算量，实现也更简单。近端策略优化能够使用旧策略的采样轨迹作为经验，提高了采样数据的利用率。

近端策略优化算法在上文所描述的 A2C 基础上主要改进了演员网络的优化目标，近端策略优化算法的优化目标如下：

$$A^{\pi_\theta}(s_t, a_t) = r_t + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t) \tag{3.8}$$

$$J^{\theta'}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta'}} = \left[\frac{p_{\theta'}(a_t|s_t)}{p_\theta(a_t|s_t)} A^{\theta'}(s_t, a_t)\nabla log p_\theta(a_t^n|s_t^n)\right] \tag{3.9}$$

其中，θ 是需要优化的参数，θ' 是真正与环境进行交互的用于做示范（demonstration）的策略。用 θ' 与环境进行交互采样出 s_t, a_t 之后计算 (s_t, a_t) 的优势函数 $A^{\theta'}(s_t, a_t)$。KL 散度的计算很复杂，本文使用一种广泛使用的近端策略优化算法的改进方法——近端策略优化剪裁。

$$J^{\theta'}(\theta) \approx \sum_{(s_t,a_t)} \min\left(\frac{p_{\theta'}(a_t|s_t)}{p_\theta(a_t|s_t)}A^{\theta'}(s_t,a_t), clip(\frac{p_{\theta'}(a_t|s_t)}{p_\theta(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon)\right) \tag{3.10}$$

其中，$clip$ 是指如果第一项小于第二项就输出 $1-\varepsilon$，如果第一项大于第三项就输出 $1+\varepsilon$，ε 是一个超参数，一般我们设置为 0.1。

3.6 自适应流处理系统工作流

在仿真环境中的客户端上，在每隔 T 个单位处理时间区间，客户端中的视频流捕获设备（摄像机）从实时的视频流中抽取出相应的帧，将这些帧作为一个待发送的视频块的组成部分缓存在缓冲区队列中。然后，系统根据自适应决策算法采样出一个动作即视频编码配置以及卸载目标 $a_t = (fr, rs, q, target)$，从缓冲区提取出缓存中相应的视频帧数据，按照决策给出的配置编码为一个视频块。自适应决策算法再根据编码出的视频块信息已经对当前环境的观测进行综合决策，选择将视频块卸载到本地或是某一个边缘服务器中进行后续的任务处理。此外，如果卸载到边缘服务器上，处理后的结果需要及时回传给客户端的自适应决策算法以衡量该步过程的奖励。如若决定在本地设备处理，则可以立即获得结果，无须视频块传输和分析任务回传。在经过 T 个时间步之后，将收集到的轨迹作为新的训练数据加入到经验池中并更新策略。

4 系统实现与实验评估

4.1 系统架构设计与实现

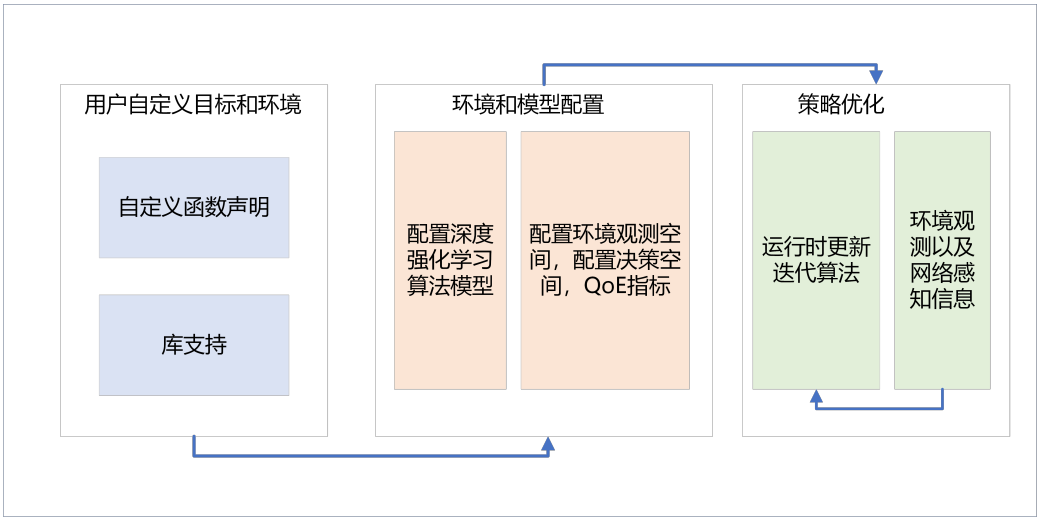

图 4.1 系统架构设计

按照3.3以及3.4中所描述的系统架构，从用户的使用角度来看，用户先通过指定3.3所描述的相应接口，包括环境，评估指标以及所使用的深度强化学习算法，然后系统将根据用户定义的 API 配置深度强化学习算法模型和优化的目标，然后系统开始在仿真环境中进行算法迭代与更新。过程如图4.1所示。

从系统的实现来看，我们将系统各个主要组成部分以及相应的核心功能抽象为独立的模块，然后在这些模块的基础之上，为系统封装一个统一的 API 接口，用于用户自定义不同的任务场景和优化目标。系统核心模块如下所示：

- 网络仿真模块：该模块实现自定义的网络仿真环境，支持真实网络环境中采样的网络带宽数据或者自定义生成的网络带宽数据，带宽单位为字节每秒 Bps。
- 边缘服务器模块：该模块实现自定义的边缘服务器，用于执行不同的处理任务。通过在边缘服务器上部署不同能力的任务处理模型，从而模拟计算资源差异性。
- 客户端模块：该模块实现自定义的客户端服务，用于生成或者从环境中采样数据，并将采样的数据以流的形式传输给边缘服务器端。
- 深度强化学习算法模块：该模块实现深度强化学习的决策算法。通过集成到客

户端模块中，可以获取客户端模块的内部状态信息以及客户端感知到的网络环境信息。

- 数据流处理模块：该模块实现对接收到的数据进行相应的处理。通过集成到客户端模块和边缘服务器端模块，处理卸载的数据和任务，同时数据流处理模块支持不同处理能力的神经网络模型。

4.2 实验设计

在这一节，我们首先介绍物联网视频流分析任务背景以及使用到的相关工具和细节。接着讨论网络仿真设计的实现，实验中分别使用了真实世界中采样的带宽数据以及自定义的网络带宽记录数据来生成模拟的网络环境用于训练和测试。最后阐述实验流程，分析各个模块的实现细节以及在实验中的具体交互过程。

4.2.1 物联网中视频流卸载任务场景设置

物联网中视频流处理的一个典型应用场景是视频流数据卸载与目标检测。客户端设备上的摄像头实时采集数据（视频帧），并缓存到摄像头设备中的缓冲区中。然后通过抽取缓冲区的多个帧并使用 H.264 或者其他编码方法编码为一个视频块，然后通过网络发送到边缘的服务器中进行目标检测，并将分析结果实时回传给数据中心。从而实现物联网视频流数据的边缘处理，缓解了大量原始视频流上传至数据中心带来的网络负载。在实验场景中，我们假设系统中的智能摄像头具备初步处理计算密集型任务的能力，同时边缘侧有多个服务器可供视频流数据卸载选择。因此，不同于以往的工作往往假定系统中只有一个可供卸载的目标，在我们的系统中，可卸载的目标包括本地以及多个边缘服务器。

为了模拟这个过程，我们使用了目标检测中常用的数据集 MOT16[28] 中的 MOT16-04 和 Udacity 自动驾驶数据集[31] 分别作为两个任务场景的视频帧数据模拟实时视频流，我们设定所使用的摄像头的采样效率为每秒 30 帧，采样的原始视频帧的分辨率为 1920x1080。其中，MOT16-04 中的物体移动速度较慢，用于模拟低速移动目标检测任务，Udacity 中的物体移动数据较快，用于模拟高速移动目标检测任务。

为了在真实网络条件下评估系统，我们创建了一个网络轨迹数据，将几个公开数据集进行了合并：我们选择了美国联邦通信委员会（FCC）提供的 2022 年 1 月的

宽带数据集[32]和挪威收集的 3G/HSDPA 移动数据集[33]。FCC 数据集包含超过 100 万个网络带宽数据，每个记录了 2100 秒内的平均吞吐量，时间粒度为 5 秒。我们从 2022 年 1 月收集的带宽数据中中随机选择，并将它们连接起来作为我们的网络带宽数据集。HSDPA 数据集包含了 30 分钟的吞吐量测量数据，使用移动设备在移动过程中（例如公交车、火车等）观看流媒体视频的带宽数据。我们使用 HSDPA 数据集中从 Ljansbakken 到 Jernbanetorget 的公交车上测量得到的网络带宽轨迹。为了避免比特率选择变得简单的情况，即总是选择最大比特率或者网络在较长时间内无法支持任何可用比特率的情况，我们只考虑平均吞吐量小于 6 Mbps 的原始网络带宽数据。此外，基于以上带宽我们还合成了两段带宽，其中一段在前半部分时间段内带宽较低（0-3Mbps），后半段时间网络带宽较高（3Mbps-6Mbps）。另一段则相反，前半部分网络带宽较高，后半部分网络带宽较低。这两段带宽用于模拟客户端与两个边缘服务器的网络连接状况差异较大的场景。

为了较为准确的评估该系统的性能，我们使用 JetStream 方法[27]作为基准线。由于 JetStream 的原始论文中的实验场景与本系统之间存在较大差异，我们在 JetStream 的核心思想之上，将其调整适用于本系统的实验场景。具体而言，我们根据环境的状态，固定选取当前客户端与边缘服务器网络连接带宽中最大的作为卸载目标。根据当前最大带宽的大小和当前视频块的比值，我们在 $[0.2, 0.4, 0.6, 0.8]$ 区间内分别设置了不同的策略。

我们为基于深度强化学习方法的设定的环境观测空间包括发送上一个视频块过程的平均带宽，发送上一个视频块过程的带宽波动方差，剩余的缓冲区容量，发送的上一个视频块的大小，当前带宽，发送上一个视频块的时延。动作空间包括分辨率，帧率，量化指标和卸载目标。其中分辨率的包括 $[1920, 1080], [1600, 900], [1280, 720], [960, 540]$ 四个配置，帧率包括 $[30, 15, 10, 6, 5]$ 五个配置，量化指标参考 gstreamer 文档的推荐包括 $[23, 28, 33, 38]$ 四个配置。卸载目标包括本地，边缘服务器 1，边缘服务器 2 三个选择。因此，动作空间总共可选的动作为 240 个。

4.2.2 评估指标

不同任务场景下视频流卸载和分析任务的体验质量（QoE）存在显著差异[1-4]。在本实验的视频流卸载和任务分析场景中，我们主要考虑三个重要的指标视频目标

检测精度，传输时延以及能耗。为了有效反应系统的效能，我们综合这三个指标作为系统的体验质量评估指标。

为了评估视频流帧数据的目标检测精度，我们使用目标检测任务中常用的 mAp 指标作为评估依据。mAP（mean Average Precision）是一种常用的评估指标，通过综合考虑模型的检测准确率和召回率，能够客观地评估模型的性能，比较不同模型或算法的优劣。mAP 结合了准确率（Precision）和召回率（Recall）两个指标，通过计算不同类别的平均精度（AP）并求取其平均值得到最终的 mAP。首先，对于每个类别，通过将检测结果按照置信度从高到低进行排序，计算不同阈值下的精度和召回率。其中，精度表示模型预测为正例的样本中实际为正例的比例，召回率表示实际为正例的样本中被模型正确预测为正例的比例。然后，通过绘制精度-召回率曲线（Precision-Recall Curve），可以得到不同阈值下的精度和召回率变化情况。曲线下的面积即为该类别的平均精度（AP）。最后，对于所有类别的平均精度，取其平均值作为模型的最终 mAP。mAP 的取值范围在 0 到 1 之间，数值越大表示模型性能越好。一个高 mAP 值意味着模型在多类别目标检测中具有较高的准确率和召回率，能够更好地识别出目标并保持低误检率。但是由于系统需要为视频流配置提供决策，因此，针对不同的视频块，还需要考虑帧率的影响。

在视频流目标检测任务中，从视频中以不同的采样率提取帧，将其压缩为不同的分辨率，然后通过边缘服务器上的神经网络模型进行处理。不同的配置会导致不同的精度和能耗。由于边缘节点用作视频处理的后端，通过动态性的网络链接将视频从其客户端传输到边缘服务器所造成的时延是不可避免的。因此，有效的视频分析卸载涉及配置适应时延的问题。为了简化仿真系统的复杂度，在本系统中，时延包括以下部分：在客户端进行视频编码的时延，将编码后的视频块卸载到边缘服务器的时延，链路上传输的往返时延 RTT 以及边缘服务器上进行分析的时延。

由于物联网设备通常长时间处于在线状态，电池寿命是主要问题之一，因此在本系统中我们还考虑能耗问题。在视频流卸载任务场景中的能耗主要由两部分组成：数据传输引起的传输能耗和本地视频帧处理引起的处理能耗。传输能耗与上传到边缘服务器的数据大小成正比。具有分辨率 r 的视频帧的数据大小被计算为 αr^2 比特，其中 α 是一个常数。我们设每传输一个比特所需要的能耗为 $5 * 10^{-6}\mathcal{J}$，神经网络每

处理一帧图像所消耗的能耗为 $5\mathcal{J}/frame$。体验质量的能耗定义为：

$$processing \quad energy = \mu * frames\_num \tag{4.1}$$

$$transmission \quad energy = \alpha * \gamma * frames\_num * bits \tag{4.2}$$

$$sum \quad of \quad energy = processing \quad energy + transmission \quad energy \tag{4.3}$$

综上所述，系统在物联网视频流卸载和目标检测任务中的体验质量指标（QoE）定义如下：

$$QoE = mAps \times framerate \times \alpha - delay \times \beta - energy/norm \tag{4.4}$$

4.3 实验结果分析

在本节中，我们将探讨实验组的设置，并对实验结果进行分析。考虑到现有工作与本系统的场景存在较大差异，因此我们在现有工作的核心思想基础上，将其提出的方法调整适用于本系统的实验场景。具体而言，我们以 JetStream[27] 作为基准线，通过将网络环境分为真实网络环境和自定义的网络带宽环境，使得这两种实验环境具有较大的网络状况差异。我们将视频流分析任务划分为高速移动目标检测和低速移动目标检测两个任务，并分别比较了基于深度强化学习方法和基准线的效果。在每组实验中，我们分别发送 1000 个视频块，记录发送每个视频块区间的平均网络带宽，对每个视频块进行目标检测的平均 mAP，卸载决策和视频编码配置等信息。策略的动作空间包括分辨率，量化指标，帧率以及卸载目标。其中，可选的分辨率包括 $[1920p, 1600p, 1280p, 960p]$；使用 gstreamer 对视频帧进行编码时可选的配置为 $[18, 23, 28, 33, 38, 43]$；帧率可选的配置为 $[30, 15, 10, 6, 5]$；卸载目标包括 $[0, 1, 2]$，0 表示本地设备，1 和 2 分别表示不同的边缘服务器。以上四个配置的任意组合即为策略所有可能产生的决策配置。为了平衡用户体验质量（QoE），实验中综合了目标检测精度，时延以及能耗三个指标。实验中具体设置为 $QoE = \alpha * accuracy - \beta * delay - \gamma * energy$，其中，$\alpha$，$\beta$，$\gamma$ 为三个可调节的超参数，不同的权重用于调节三个指标的重要性并且实现对 QoE 的归一化。在边缘服务器上我们使用 YOLOV5M[34] 目标检测模型或者 facebook 提出的目标检测模型 DETR[35]。实验过程中，我们使用性能最优的模型 YOLOV5X 预先对模拟实时视频流的数据集进行分析，并将每帧的结果记录下来，作为实验的基准，从而加快决策模型的训练速度。具体的实验参数配置和结果分析

如下。

4.3.1 深度强化学习方法与传统方法对比分析

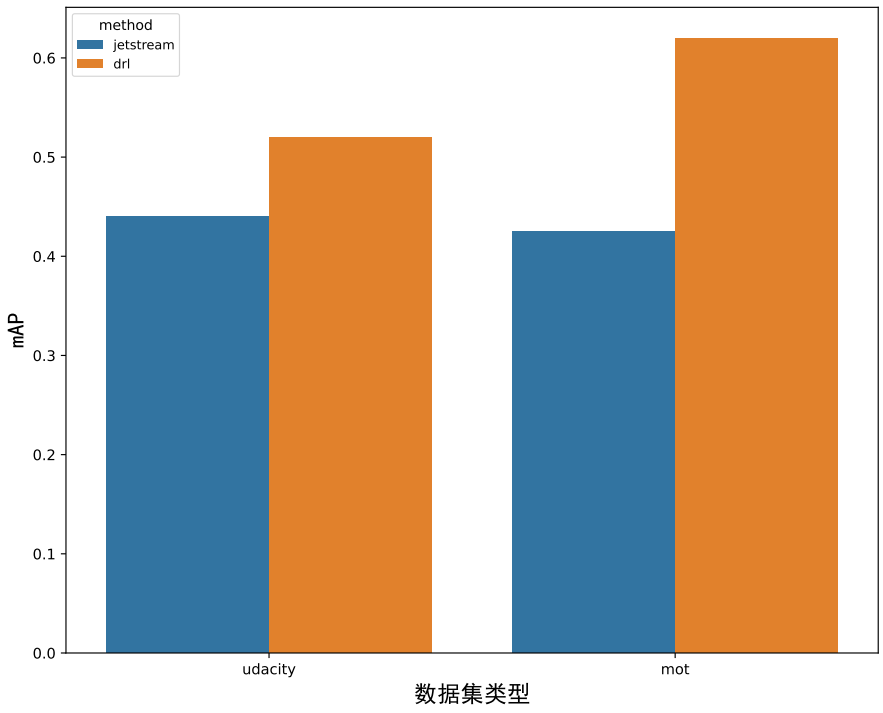

图 4.2 真实网络带宽环境下，不同视频流内容的目标检测 mAP

图4.2比较了不同视频流内容下，基于深度强化学习算法的方法和基于 JetStream 的方法的性能。为了公平性，我们在边缘服务器 1 和边缘服务器 2 上均使用的 YOLOV5M[34]模型。图4.2横坐标为模拟视频流所使用的数据集，纵坐标为在本组实验中对所有发送视频块进行目标检测获得的平局 mAP。可以发现，在 MOT16-03 作为视频流的任务中，基于深度强化学习的方法的平均 mAP 与基于 JetStream 的方法的平均 mAP 相比提高了 14.4%；在以 udacity 为视频流的任务中，基于深度强化学习方法的平均 mAP 相比于基于 JetStream 方法的平均 mAP 提高了 21.9%。在两个任务中基于深度强化学习的方法在 mAP 上均有较明显的提升。图4.3比较了不同视频流内容下，基于深度强化学习算法的方法和基于 JetStream 的方法的性能比较。为了公平性，我们在边缘服务器 1 和边缘服务器 2 上均使用的 YOLOV5M[34]模型。可以发现，在 MOT16-03 作为视频流的任务中，基于深度强化学习的方法的平

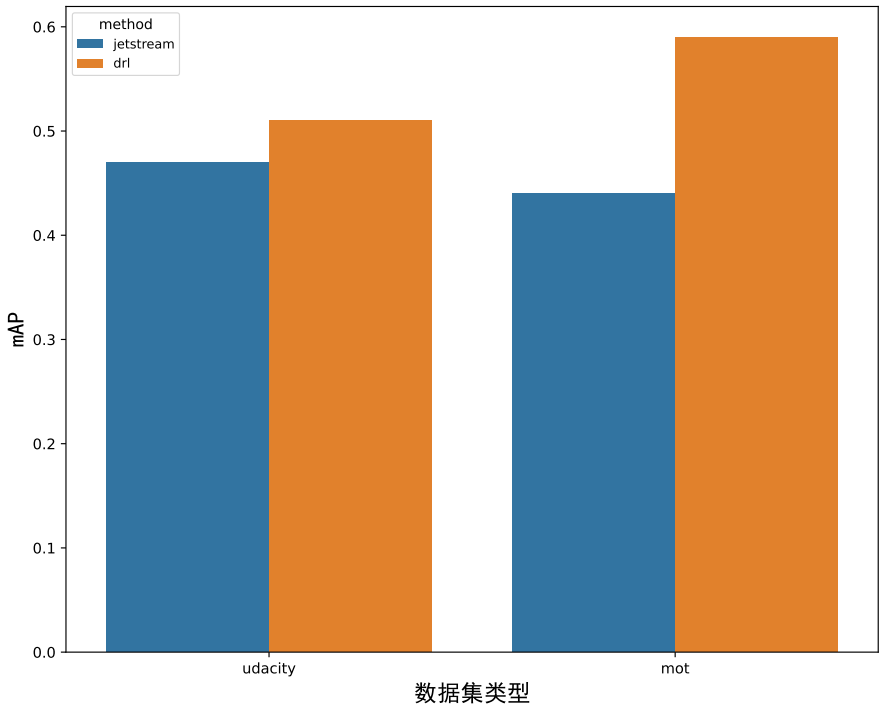

图 4.3　自定义网络环境下，不同视频流内容的目标检测 mAP

均 mAP 与基于 JetStream 的方法的平均 mAP 相比提高了 8.5%；在以 udacity 为视频流的任务中，基于深度强化学习方法的平均 mAP 与基于 JetStream 方法的平均 mAP 相比 47.5%。在客户端与不同服务器之间的网络环境存在较大差异的情况下，本系统基于深度强化学习的决策模型依然可以维持较高的精度。

为了研究深度强化学习算法模型在不同网络带宽下做的决策是否合理，我们可以通过配置 α，β，γ 三个可调节的超参数，来改变时延，目标检测精度和能耗三者的重要性。

4.3.2　时延优先策略实验结果分析

在时延优先的实验中，我们分别将 α 设置为 0.5，β 设置为 2，γ 设置为 $1*10^{-10}$。此外，为了在训练过程中强调时延的重要性，在发送上一个视频块时，如果发送过程中所选的卸载目标的平均带宽高于另一条链路，则会给予 40 的奖励值。接着，我们通过使用一段从 HDPSA 中手动选取的部分网络带宽数据进行组合，生成了两段网络带宽数据，分别作为客户端与两个边缘服务器之间的网络状况。然后，我们在仿真环境中测试发送 1000 个视频，观察每一个视频块的目标检测精度以及时延。如

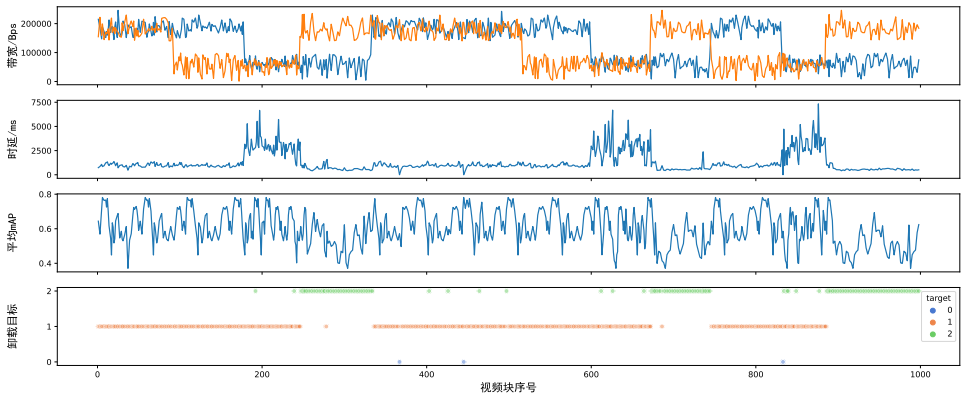

图 4.4　时延优先策略下，随着视频流卸载处理，网络带宽状况、决策、时延以及目标检测精度 mAP 的变化

图4.4中间所示，带宽状况包括两段网络状况均良好，两段网络状况的优劣交替以及两段网络状况均处于低带宽的情况。实验结果如图4.4所示。图4.4横坐标为每个视频块的序号，第一幅图的左侧纵坐标为发送每个视频块的时延。第二幅图纵坐标为每个视频块发送期间，客户端与边缘服务器 1 和边缘服务器 2 之间网络连接的平均带宽。第三幅图的纵坐标为每个视频块经过目标检测模型分析后的平均 mAP，用于反应任务分析的处理性能。第四幅图的纵坐标为深度强化学习策略为每一个视频生成的卸载目标。

我们以序号 0-300 的视频块发送过程为例，可以看出在序号为 0-100 的视频块中，客户端与边缘服务器 1，2 之间的网络状况均处于良好状态，同时卸载策略也维持了较低的发送时延。在序号为 100-200 的视频块发送区间内，由于与边缘服务器 2 之间的网络状况变差，生成的深度强化学习策略选择了将视频块卸载到边缘服务器 1，从而继续维持了较低的时延。在序号为 200-230 的视频块发送区间内，两个网络连接的带宽均较低，此时，视频块发送的时延表现出升高的趋势。在序号 230-300 的视频块发送区间内，客户端与边缘服务器 1 之间的网络状况较差，而与边缘服务器 2 之间的网络带宽较好，此时策略更多的选择将视频块卸载到边缘服务器 2 上，维持了较低的时延。

4.3.3　精度优先策略实验结果分析

在某些场景下，我们可能更关注视频目标检测的准确度，而对于时延有较高的容忍。在准确度优先的实验中，我们部署在边缘服务器 1 上的目标检测模型效果由

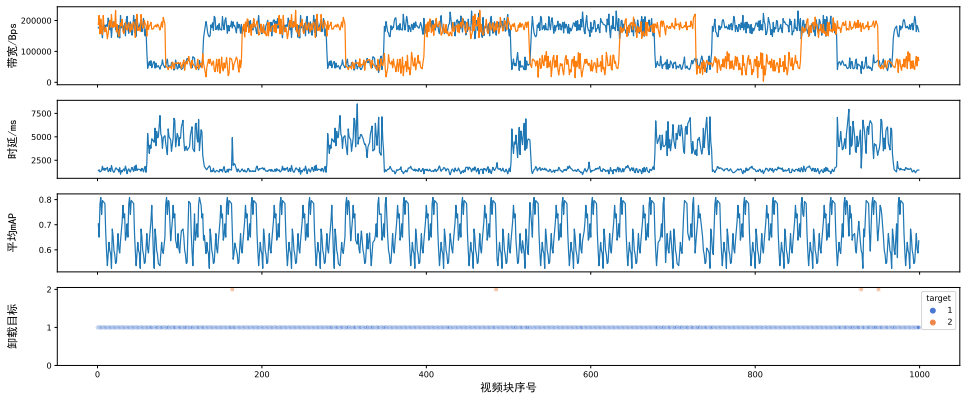

图 4.5　精度优先策略下，随着视频流卸载处理，网络带宽状况、决策、时延以及目标检测精度的变化

于在边缘服务器 2 上的模型。重要性权重方面，我们分别将 α 设置为 2，β 设置为 2，γ 设置为 $1 * 10^{-10}$。此外，为了防止大量视频块卸载到本地从而对本地设备造成较大负载，在奖励设计中，策略做出卸载到本地的决策会得到 -5 的奖励。实验结果如图4.5所示。

与时延优先的策略相比，综合发送的所有 1000 个视频块的结果，精度优先策略的平均 mAP 为 0.662，而时延优先策略的平均 mAP 为 0.608，提升为 8%。但是在时延方面，精度优先策略每个视频块发送平均时延为 2380 毫秒，而时延优先策略的发送每个视频块的平均时延为 1445 毫秒。此外，由于边缘服务器 1 的计算资源更加丰富，训练得到的深度强化学习模型所做出的决策几乎都选择边缘服务器 1 作为卸载目标。为了平衡缓冲区的使用，算法则是通过选择不同分辨率，帧率和量化指标的配置，从而减少缓冲区溢出而导致的实时视频块丢失等问题。

4.3.4　模型训练过程优化

在较早的实现过程中，系统获得对环境观测之后，实时从视频帧数据集中选取相应的帧，并实时进行编码压缩，然后再发送到边缘服务器端。这个过程需要等待 gstreamer 对视频帧编码压缩，严重降低了对环境的采样效率以及训练速度。在实测中，系统训练 15000 步需要大致 7 个小时。此外，实时的编码压缩可能会导致分析结果目标检测分析结果波动，对训练的稳定性造成较大影响。为了优化训练稳定性问题，同时提高实验的采样效率，我们使用本次实验中效果最好的 YOLOV5X 模型预先对视频帧数据集进行目标检测，将每一帧的检测结果记录在表中。然后将视频帧划

分为固定的视频块，预先使用不同推理能力的模型（包括 YOLOV5M，YOLOV5N）和 gstreamer 对所有视频块和所有可组合的配置进行处理，将处理结果记录到一个表格 *gt_table* 中。在此之后的实验中，我们按照每次所生成的决策，在表中查询相应的结果，大大加速了训练过程并且显著提高了有效性。在实测中，我们采样 750000 次，仅需 30 分钟左右。

结论

在本次毕业设计中，我们设计并实现了一个基于深度强化学习的物联网自适应流处理系统。该系统通过提供将底层实现模块进行封装，在上层为用户提供一个统一的 API，使得用户能够根据自定义的任务环境和体验质量目标，对不同场景下的物联网流处理任务进行优化。在实验环节，我们基于物联网中的典型的视频流处理任务构建了一个仿真环境，并综合视频分析精度，时延和能耗三个指标作为体验质量，使用系统对视频块卸载任务进行优化。相比 JetStream 等传统手动配置策略，实现了精度，时延，能耗和网络带宽状况以及不同计算能力之间更优的平衡，提高了计算资源和网络资源的利用率，有效缓解了网络资源的稀缺性和动态性带来的影响。同时，我们将场景推广到环境中存在多个边缘服务器以及本地设备也可对计算密集型任务进行初步处理，进一步扩大了该系统的使用范围。本文的具体工作内容如下：

- 本文通过对物联网的基本组成部件进行抽象封装，在上层为系统提供了一个接口，使用户能够针对不同的物联网流处理场景快速搭建流处理系统。
- 相对于传统的手动配置或者基于最优化方法的策略，本系统引入基于深度强化学习的方法，简化了复杂物联网环境的建模并且能够获得更优的流处理策略。
- 本系统对现有工作的任务场景进行了扩展，支持将任务卸载到多个边缘服务器或者任务本地进行处理，扩展了流处理框架的使用范围。

但是本文依然存在一些不足，针对物联网流处理应用面临的网络资源高度动态、实时协同困难，资源利用的最优化问题以及将该系统实用化还有许多挑战亟须解决，我们计划在未来从以下几方面对其进行改进。

- 系统的有效性。如何设计深度强化学习的奖励函数是整个领域都还需深入研究的问题，在物联网流处理系统的场景中，存在较多需要考虑的指标，包括流处理效果，时延，能耗，资源利用率等，如何设计一个合理的体验质量指标具有一定的挑战。由于深度强化学习内部机制在很大程度上是一个 “黑盒”，量化指标的细微变化都有可能引起模型策略产生较大的变化。已有部分工作对某些任务场景的 QoE 指标进行了深入研究，但是如何设计一个更加鲁棒的指标，仍需深入研究。

- 系统的实用性。现有流处理系统，如 FLink[6]，Spark[5]等已经大规模部署。但是这些框架并不能在物联网设备和场景中使用，如何将基于深度强化学习的流处理框架结合到现有流行的框架中，使得其能够被工业领域使用，仍需深入研究和开发。
- 系统的可扩展性。随着 AI，5G 以及嵌入式的高速发展，未来的物联网中的数据量和接入设备将进一步增长，物联网中的各类设备的计算资源也会变得更加丰富将给物联网的研究带来更大的挑战。虽然本系统的一定程度上对场景和设备扩展性问题做了初步探索，但是在更加实际的问题场景中，接入设备往往是动态的。如何在设备动态接入的物联网环境中，实现流处理决策的优化这一问题，我们在未来需要进一步开展工作。

参考文献

[1] ZHANG M, WANG F, LIU J. Casva: Configuration-adaptive streaming for live video analytics[C/OL]//IEEE INFOCOM 2022 - IEEE Conference on Computer Communications. 2022: 2168-2177. DOI: 10.1109/INFOCOM48880.2022.9796875.

[2] WANG C, ZHANG S, CHEN Y, et al. Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics[C/OL]//IEEE INFOCOM 2020 - IEEE Conference on Computer Communications. 2020: 257-266. DOI: 10.1109/INFOCOM41043.2020.9155524.

[3] MAO H, NETRAVALI R, ALIZADEH M. Neural adaptive video streaming with pensieve[C/OL]//SIGCOMM '17: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. New York, NY, USA: Association for Computing Machinery, 2017: 197–210. https://doi.org/10.1145/3098822.3098843.

[4] ZHANG B, JIN X, RATNASAMY S, et al. Awstream: Adaptive wide-area streaming analytics[C/OL]//SIGCOMM '18: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. New York, NY, USA: Association for Computing Machinery, 2018: 236–252. https://doi.org/10.1145/3230543.3230554.

[5] ZAHARIA M, DAS T, LI H, et al. Discretized streams: Fault-tolerant streaming computation at scale[C/OL]//SOSP '13: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. New York, NY, USA: Association for Computing Machinery, 2013: 423–438. https://doi.org/10.1145/2517349.2522737.

[6] KATSIFODIMOS A, SCHELTER S. Apache flink: Stream analytics at scale[C/OL]//2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW). 2016: 193-193. DOI: 10.1109/IC2EW.2016.56.

[7] ZHANG H, ANANTHANARAYANAN G, BODIK P, et al. Live video analytics at scale with approximation and Delay-Tolerance[C/OL]//14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). Boston, MA: USENIX Association, 2017: 377-392. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang.

[8] ZHU Z, FENG X, TANG Z, et al. Power-efficient live virtual reality streaming using edge offloading[C/OL]//NOSSDAV '22: Proceedings of the 32nd Workshop on Network and Operating Systems Support for Digital Audio and Video. New York, NY, USA: Association for Computing Machinery, 2022: 57–63. https://doi.org/10.1145/3534088.3534351.

[9] STOCKHAMMER T. Dynamic adaptive streaming over http –: Standards and design principles[C/OL]//MMSys '11: Proceedings of the Second Annual ACM Conference on Multimedia Systems. New York, NY, USA: Association for Computing Machinery, 2011: 133–144. https://doi.org/10.1145/1943552.1943572.

[10] JIANG J, ANANTHANARAYANAN G, BODIK P, et al. Chameleon: Scalable adaptation of video analytics[C/OL]//SIGCOMM '18: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. New York, NY, USA: Association for Computing Machinery, 2018: 253–266. https://doi.org/10.1145/3230543.3230574.

[11] HAZELCAST. Hazelcast，real-time stream processing platform[C].

[12] GOOGLE. Google nest cam[C/OL]//2017. https://store.google.com/us/category/nest_cams?hl=en-US&GoogleNest&pli=1.

[13] NETATMO. Netatmo[C/OL]//2017. https://www.netatmo.com/en-gb.

[14] AMAZON. Aws iot greengrass[C/OL]//https://azure.microsoft.com/en-us/services/iot-edge.

[15] OUSTERHOUT K, WENDELL P, ZAHARIA M, et al. Sparrow: Distributed, low latency scheduling[C/OL]//SOSP '13: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. New York, NY, USA: Association for Computing Machinery, 2013: 69–84. https://doi.org/10.1145/2517349.2522716.

[16] ZAKI Y, PöTSCH T, CHEN J, et al. Adaptive congestion control for unpredictable cellular networks[C/OL]//SIGCOMM '15: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. New York, NY, USA: Association for Computing Machinery, 2015: 509–522. https://doi.org/10.1145/2785956.2787498.

[17] SUN Y, YIN X, JIANG J, et al. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction[C/OL]//SIGCOMM '16: Proceedings of the 2016 ACM SIGCOMM Conference. New York, NY, USA: Association for Computing Machinery, 2016: 272–285. https://doi.org/10.1145/2934872.2934898.

[18] WINSTEIN K, SIVARAMAN A, BALAKRISHNAN H. Stochastic forecasts achieve high throughput and low delay over cellular networks[C]//nsdi'13: Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation. USA: USENIX Association, 2013: 459–472.

[19] HUANG T, ZHOU C, ZHANG R X, et al. Comyco: Quality-aware adaptive video streaming via imitation learning[C/OL]//MM '19: Proceedings of the 27th ACM International Conference on Multimedia. New York, NY, USA: Association for Computing Machinery, 2019: 429–437. https://doi.org/10.1145/3343031.3351014.

[20] WANG H, WU K, WANG J, et al. Rldish: Edge-assisted qoe optimization of http live streaming with reinforcement learning[C/OL]//IEEE INFOCOM 2020 - IEEE Conference on Computer Communications. 2020: 706-715. DOI: 10.1109/INFOCOM41043.2020.9155467.

[21] PANG H, ZHANG C, WANG F, et al. Towards low latency multi-viewpoint 360° interactive video: A multimodal deep reinforcement learning approach[C/OL]//IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. 2019: 991-999. DOI: 10.1109/INFOCOM.2019.8737395.

[22] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing atari with deep reinforcement learning[A]. 2013. arXiv: 1312.5602.

[23] WILLIAMS R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning[M/OL]. Boston, MA: Springer US, 1992: 5-32. https://doi.org/10.1007/978-1-4615-3618-5_2.

[24] MNIH V, BADIA A P, MIRZA M, et al. Asynchronous methods for deep reinforcement learning[A]. 2016. arXiv: 1602.01783.

[25] SCHULMAN J, LEVINE S, ABBEEL P, et al. Trust region policy optimization[C]// International conference on machine learning. PMLR, 2015: 1889-1897.

[26] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal policy optimization algorithms[A]. 2017. arXiv: 1707.06347.

[27] RABKIN A, ARYE M, SEN S, et al. Aggregation and degradation in jetstream: Streaming analytics in the wide area[C]//NSDI'14: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. USA: USENIX Association, 2014: 275–288.

[28] MILAN A, LEAL-TAIXE L, REID I, et al. Mot16: A benchmark for multi-object tracking [A]. 2016. arXiv: 1603.00831.

[29] WENG J, CHEN H, YAN D, et al. Tianshou: A highly modularized deep reinforcement learning library[J/OL]. Journal of Machine Learning Research, 2022, 23(267): 1-6. http://jmlr.org/papers/v23/21-1127.html.

[30] SHEN B, CAO C, LIU T, et al. Neural adaptive iot streaming analytics with rl-adapt [C/OL]//2021 17th International Conference on Mobility, Sensing and Networking (MSN). 2021: 382-389. DOI: 10.1109/MSN53354.2021.00065.

[31] UDACITY. Udacity self-driving car dataset[C/OL]//2021. DOI: https://github.com/udacity/self-driving-car.

[32] COMMISSION F C. Measuring broadband raw data releases - fixed[EB/OL]. 2022. https://www.fcc.gov/oet/mba/raw-data-releases#dataTable.

[33] RIISER H, VIGMOSTAD P, GRIWODZ C, et al. Commute path bandwidth traces from 3g networks: Analysis and applications[C/OL]//MMSys '13: ACM. New York, NY, USA: Association for Computing Machinery, 2013: 114–118. https://doi.org/10.1145/2483977.2483991.

[34] ULTRALYTICS. ultralytics yolov5[EB/OL]. 2023. https://github.com/ultralytics/yolov5.

[35] CARION N, MASSA F, SYNNAEVE G, et al. End-to-end object detection with transformers[J/OL]. Lecture Notes in Computer Science, 2020: 213–229. http://dx.doi.org/10.1007/978-3-030-58452-8_13.

附录 A　项目代码

A.1　API 代码文件

```
from gymnasium import spaces
import gymnasium
from tianshou import policy

class AdaptationPipeline():
    def __init__(self) -> None:
        self.env = None
        self.config_set = None
        self.algorithm = None

    def set_config_set(self, configurations):
        self.config_set = configurations
        self.env.action_spaces = spaces.Discrete(len(configurations))

    def set_env(self, env: gymnasium.Env):
        self.env = env

    def set_qoe_metric(self, qoe_metric):
        self.env._get_reward = qoe_metric

    def set_algorithm(self, algorithm: policy.BasePolicy):
        self.rl_policy = algorithm
```

A.2　仿真环境代码文件

```
import gymnasium
from gymnasium import spaces
import numpy as np
from typing import List
from sim.client import Client
from sim.server import Server
from sim.util import energy_consuming
import pandas as pd
import math
import time
from collections import OrderedDict

# ================hyperparameter=========================
BUFFER_SIZE=1e6 # bytes
BYTES_IN_MB = 1e6
MILLS_PER_SECOND = 1e3
BW_NORM = 1e4

# action space
RESOLUTION = [[1920, 1080], [1600, 900], [1280, 720], [960, 540]]
```

```
QUANTIZER = [23, 28, 33, 38]
SKIP = [0, 1, 2, 4, 5]
FRAMERATE = [30, 15, 10, 6, 5]
rs_mapping = {1920: [1920, 1080], 1600:[1600,900], 1280: [1280,720],
    960:[960, 540]}
# other
LOG = "log/"

# ================================================

class SimEnv(gymnasium.Env):
    metadata = {"render_modes": ["human"]}

    def __init__(self, algo:str) -> None:
        self.client = Client("yolov5n", BUFFER_SIZE)
        self.server1 = Server("yolov5m", 1)
        self.server2 = Server("yolov5x", 2)
        self.rtt = 80
        self.targets = [self.client, self.server1, self.server2]
        # stream
        self.drain_mode = False

        # env
        self.actions_mapping = self.__build_actions_mapping(3)
        self.action_space =
            spaces.Discrete(len(self.actions_mapping))
        self.observation_space = spaces.Dict({
            "past_bws_mean": spaces.Box(low=np.array([0, 0]),
                high=np.array(self.server1.network.get_max_bw())/BYTES_IN_MB,
                dtype=np.float32),
            "past_bws_std": spaces.Box(low=np.array([0, 0]),
                high=np.array(self.server1.network.get_max_bw())/BYTES_IN_MB,
                dtype=np.float32),
            "available_buffer_size": spaces.Box(low=-5 *
                BUFFER_SIZE/ BYTES_IN_MB, high=5 * BUFFER_SIZE /
                BYTES_IN_MB, dtype=np.float32),
            "segment_size": spaces.Box(low=0, high=2.0 * BUFFER_SIZE
                / BYTES_IN_MB, dtype=np.float32),
            "current_bws": spaces.Box(low=np.array([0, 0]),
                high=np.array(self.server1.network.get_max_bw())/BYTES_IN_MB,
                dtype=np.float32),
            "past_segment_delay": spaces.Box(low=0, high=np.inf,
                dtype=np.float32)
        })
        # log
        self.log = LOG + algo + \
            time.strftime("%Y-%m-%d-%H:%M:%S", time.localtime()) +
                ".csv"
        self.training_log = LOG + algo + \
            time.strftime("%Y-%m-%d-%H:%M:%S", time.localtime()) +
                "-train.csv"


    def __build_actions_mapping(self, links) -> List:
```

```
        configs = []
        for resolution in RESOLUTION:
            for q in QUANTIZER:
                for framerate in FRAMERATE:
                    for id in range(links):
                        configs.append(
                            {"rs": resolution, "fr": framerate, "q":
                                q, "target": id})
        return configs



    def update_state(self, config):
        if self.client.empty():
            # retrieve one segment
            seg_idx, seg_size = self.client.retrieve_segment(config)
            bws = [[self.server1.network.step()],
                [self.server2.network.step()]]
            return {"empty": True, "drain_mode": self.drain_mode,
                "mAps": 0,
                    "transmission_delay": 0, "capture_segments": 1,
                    "bws_mean1": np.mean(bws[0]), "bws_mean2":
                        np.mean(bws[1]),
                    "bws_std1": 0, "bws_std2": 0, "segment_index":
                        seg_idx,
                    "frames_num": config["fr"], "segment_size":
                        seg_size,
                    "current_bw1": self.server1.network.current_bw,
                        "current_bw2":
                        self.server2.network.current_bw,
                    "resolution": config["rs"], "framerate":
                        config["fr"], "quantizer": config["q"],
                    "target": config["target"],
                        "available_buffer_size":
                        self.client.get_buffer_vacancy(),
                    "processing_energy": 0, "transmission_energy": 0}

        target = self.targets[config["target"]]
        seg_idx, seg_size, seg_config = self.client.pop_buffer()
        _, _, mAps = target.get_gt(seg_config["rs"],
            seg_config["q"], seg_config["fr"], seg_idx)
        # compute transmission delay if sent to remote server
        bws = [[], []]
        transmission_delay = 0
        to_sent_seg_size = seg_size
        cap_segs = 0
        if config["target"] != 0:
            while to_sent_seg_size > 0:
                current_bws = [self.server1.network.step(),
                    self.server2.network.step()]
                bws[0].append(current_bws[0])
                bws[1].append(current_bws[1])
                bw = current_bws[config["target"] - 1]
                if bw < to_sent_seg_size:
```

```
                transmission_delay += 1000 + self.rtt
            else:
                transmission_delay += (to_sent_seg_size / bw *
                    1000 + self.rtt)
                to_sent_seg_size -= bw
        cap_segs = math.floor(transmission_delay / 1000)

        for _ in range(cap_segs):
            self.client.retrieve_segment(config, self.drain_mode)
    else:
        bws[0].append(self.server1.network.current_bw)
        bws[1].append(self.server2.network.current_bw)
    processing_energy, transmission_energy =
        energy_consuming(seg_config["fr"],
        rs_mapping[seg_config["rs"][0]], config["target"] == 0)
    if self.client.full() and self.drain_mode == False:
        self.drain_mode = True
    if self.client.empty() and self.drain_mode:
        self.drain_mode = False
    return {"empty": False, "drain_mode": self.drain_mode,
        "mAps": mAps,
                "transmission_delay": transmission_delay,
                    "capture_segments": cap_segs,
                "bws_mean1": np.mean(bws[0]), "bws_mean2":
                    np.mean(bws[1]),
                "bws_std1": np.std(bws[0]), "bws_std2":
                    np.std(bws[1]), "segment_index": seg_idx,
                "frames_num": seg_config["fr"], "segment_size":
                    seg_size,
                "current_bw1": self.server1.network.current_bw,
                    "current_bw2":
                    self.server2.network.current_bw,
                "resolution": config["rs"], "framerate":
                    config["fr"], "quantizer": config["q"],
                "target": config["target"],
                    "available_buffer_size":
                    self.client.get_buffer_vacancy(),
                "processing_energy": processing_energy,
                    "transmission_energy": transmission_energy}


def _get_obs(self, state, config):
    obs = {"past_bws_mean": np.array([state["bws_mean1"] /
        BW_NORM, state["bws_mean2"] / BW_NORM]),
            "past_bws_std": np.array([state["bws_std1"] /
                BW_NORM, state["bws_std2"] / BW_NORM]),
            "available_buffer_size":
                np.array([self.client.get_buffer_vacancy() /
                BYTES_IN_MB]),
            "segment_size": np.array([state["segment_size"] /
                BYTES_IN_MB]),
            "current_bws":
                np.array([self.server1.network.current_bw /
                BW_NORM, self.server2.network.current_bw /
```

```
                BW_NORM]),
            "past_segment_delay":
                np.array([state["transmission_delay"] /
                MILLS_PER_SECOND])}
        return obs

    def _get_reward(self, state, config):
        # Latency-first reward setting
        # alpha = 0.5
        # beta = 2
        # gamma = 1e-10
        # rew = state["mAps"] * state["framerate"] * alpha -
            state["transmission_delay"] / MILLS_PER_SECOND * beta - \
        #     (state["processing_energy"] +
            state["transmission_energy"]) * gamma
        # # print("*" * 20)
        # # print("map", state["mAps"] * state["framerate"])
        # # print("delay", state["transmission_delay"] /
            MILLS_PER_SECOND)
        # # print("energy", (state["processing_energy"] +
            state["transmission_energy"]) / 1e10)
        # # print("*" * 20)
        # if state["empty"]:
        #     return rew - 5
        # if config["target"] == 0:
        #     rew -= 5
        # else:
        #     target = config["target"]
        #     current_bws = [state["bws_mean1"],state["bws_mean2"]]
        #     if int(target - 1) ==
            current_bws.index(max(current_bws)):
        #         rew += 40
        # return rew
        # Accuracy First reward setting
        alpha = 2
        beta = 2
        gamma = 1e-10
        rew = state["mAps"] * state["framerate"] * alpha -
            state["transmission_delay"] / MILLS_PER_SECOND * beta - \
             (state["processing_energy"] +
                state["transmission_energy"]) * gamma
        # print("*" * 20)
        # print("map", state["mAps"] * state["framerate"])
        # print("delay", state["transmission_delay"] /
            MILLS_PER_SECOND)
        # print("energy", (state["processing_energy"] +
            state["transmission_energy"]) / 1e10)
        # print("*" * 20)
        if state["empty"]:
            return rew - 5
        if config["target"] == 0:
            rew -= 5
        # else:
        #     target = config["target"]
```

```
        #     current_bws = [state["bws_mean1"],state["bws_mean2"]]
        #     if int(target - 1) ==
            current_bws.index(max(current_bws)):
        #         rew += 40
        return rew

    def step(self, action):
        config = self.actions_mapping[action]
        state = self.update_state(config)
        obs = self._get_obs(state, config)
        rew = self._get_reward(state, config)
        truncated = self.truncated()
        terminated = self.terminated()
        log_info = dict(**state)
        train_info = dict(**obs, reward=rew,
                          terminated=(terminated or truncated))
        log_info_df = pd.DataFrame([log_info])
        train_info_df = pd.DataFrame([train_info])
        log_info_df.to_csv(f"{self.log}", mode='a', header=None,
            index=False)
        train_info_df.to_csv(f"{self.training_log}",
                             mode='a', header=None, index=False)
        return obs, rew, terminated, truncated, {}


    def truncated(self):
        return False

    def terminated(self):
        return self.client.sent_segs_num > 500

    def reset(self, seed=None, options=None):
        self.client.reset()
        self.server1.reset()
        self.server2.reset()
        self.drain_mode = False
        obs = OrderedDict()
        obs["past_bws_mean"] = np.array([0.0, 0.0], dtype=np.float32)
        obs["past_bws_std"] = np.array([0.0, 0.0], dtype=np.float32)
        obs["available_buffer_size"] = np.array(
            [self.client.get_buffer_vacancy() / BYTES_IN_MB],
                dtype=np.float32)
        obs["segment_size"] = np.array([0], dtype=np.float32)
        obs["current_bws"] =
            np.array([self.server1.network.current_bw,
            self.server2.network.current_bw], dtype=np.float32)
        obs["past_segment_delay"] = np.array([0], dtype=np.float32)
        return obs, {}
```

A.3 客户端代码文件

```
import os
```

```
import pandas as pd

keys = ["seg_idx", "seg_size", "mAps"]
TABLES = "tables"

class Client():
    def __init__(self, model_type, buffer_size) -> None:
        self.model_type = model_type
        self.path = TABLES + "/" + self.model_type
        self.gt = self.init_tables()
        # buffer
        self.max_buffer_size = buffer_size # bytes
        self.used_buffer_size = 0
        self.seg_idx = 0
        self.to_be_sent_segs = []
        self.droped_segs = 0
        self.sent_segs_num = 0
        self.segs_num = 35

    def init_tables(self):
        files = os.listdir(self.path)
        gt = {}
        for file in files:
            # rs, q, fr = file[:-4].split('_')
            data = pd.read_csv(f"{self.path}/{file}", header=None,
                names=keys)
            gt[file[:-4]] = data
        return gt

    def get_gt(self, rs, q, fr, seg_idx):
        """
        @return:
            seg_idx, seg_size, mAps
        """
        gt_by_config = self.gt[f"{rs[0]:04d}_{q:02d}_{fr:02d}"]
        return gt_by_config[gt_by_config["seg_idx"] ==
            seg_idx].values.tolist()[0]

    def empty(self):
        return self.used_buffer_size == 0

    def full(self):
        return self.used_buffer_size >= self.max_buffer_size

    def get_buffer_vacancy(self):
        return self.max_buffer_size - self.used_buffer_size


    def retrieve_segment(self, config, drain_mode=False):
        if self.full() or drain_mode:
            self.seg_idx = self.seg_idx % self.segs_num + 1
            self.droped_segs += 1
            return
        self.seg_idx = self.seg_idx % self.segs_num + 1
```

```
        seg_idx, seg_size, _ = self.get_gt(config["rs"],
            config["q"], config["fr"], self.seg_idx)
        assert seg_idx == self.seg_idx, "index doesn't match"
        self.used_buffer_size += seg_size
        self.to_be_sent_segs.insert(0, [seg_idx, seg_size, config])
        return seg_idx, seg_size

    def pop_buffer(self):
        seg_idx, seg_size, config = self.to_be_sent_segs.pop()
        self.used_buffer_size -= seg_size
        self.sent_segs_num += 1
        return seg_idx, seg_size, config


    def reset(self):
        self.used_buffer_size = 0
        self.seg_idx = 0
        self.to_be_sent_segs = []
        self.droped_segs = 0
        self.sent_segs_num = 0
        self.segs_num = 35
```

A.4 边缘服务器代码文件

```
import os
import pandas as pd
from sim.network import Network

keys = ["seg_idx", "seg_size", "mAps"]

TABLES = "tables"
class Server():
    def __init__(self, model_type:str, id, trace = "norway") -> None:
        self.model_type = model_type
        self.trace = trace
        self.path = TABLES + "/" + self.model_type
        self.gt = self.init_tables()
        self.network = Network(self.trace, id)

    def init_tables(self):
        files = os.listdir(self.path)
        gt = {}
        for file in files:
            # rs, q, fr = file[:-4].split('_')
            data = pd.read_csv(f"{self.path}/{file}", header=None,
                names=keys)
            gt[file[:-4]] = data
        return gt

    def get_gt(self, rs, q, fr, seg_idx):
        gt_by_config = self.gt[f"{rs[0]:04d}_{q:02d}_{fr:02d}"]
```

```
        return gt_by_config[gt_by_config["seg_idx"] ==
            seg_idx].values.tolist()[0]

    def reset(self):
        self.network.reset()
```

A.5 网络仿真代码文件

```
import numpy as np
import os
import random
TRACE = {"norway": "norway"}


class Network():
    def __init__(self, trace="norway", id=1) -> None:
        self.traces_path = TRACE[trace]
        self.time_step = 0
        self.elapsed_bws = []
        self.bws = []
        self.id = id
        if trace == "fcc":
            self.init_fcc_traces(self.traces_path)
            self.bw_id = random.randint(0, len(self.bws)-1)
            self.bw = self.bws[self.bw_id]
        elif trace == "norway":
            self.init_norway_trace(self.traces_path)
            self.bw = random.sample(self.bws, 2000)
            synthetic_bws_lower = random.sample([
                bw for bw in self.bws if bw < 100000], 300)
            synthetic_bws_higher = random.sample([
                bw for bw in self.bws if bw > 140000], 300)
            if self.id == 1:
                self.bw = [*synthetic_bws_higher,
                    *synthetic_bws_lower]
            elif self.id == 2:
                self.bw = [*synthetic_bws_lower,
                    *synthetic_bws_higher]

        self.bw = np.roll(self.bw, random.randint(
            0, len(self.bw) - 1)).tolist()
        self.current_bw = 0

    def next_bw(self):
        self.time_step = (self.time_step + 1) % len(self.bw)
        self.current_bw = self.bw[self.time_step]
        self.elapsed_bws.append(self.current_bw)
        return self.current_bw

    def get_current_bw(self):
        return self.current_bw
```

```
    def get_max_bw(self):
        return max(self.bw)

    def init_fcc_traces(self, fcc_cooked_path):
        """fcc network trace."""
        files = os.listdir(fcc_cooked_path)
        for file in files:
            trace = []
            with open(f"{fcc_cooked_path}/{file}", 'r') as f:
                lines = f.readlines()
                for bw in lines:
                    trace.append(int(bw))
            self.bws.append(trace)

    def init_norway_trace(self, traces_path):
        """norway hsdpa bandwidth logs."""
        files = os.listdir(traces_path)
        for file in files:
            with open(traces_path + "/" + file, 'r') as f:
                lines = f.readlines()
                for line in lines:
                    line = line.strip().split(' ')
                    bw = int(line[4]) / int(line[5]) * 1000  # bytes
                        per second
                    self.bws.append(int(bw / 3))

    def step(self):
        return self.next_bw()

    def reset(self):
        self.time_step = 0
        self.current_bw = 0
        self.elapsed_bws = []
        self.bw = np.roll(self.bw, random.randint(
            0, len(self.bw) - 1)).tolist()
```

A.6 强化学习算法代码文件

```
import tianshou as ts
from tianshou.data.batch import Batch
from tianshou.utils.net.common import Net, ActorCritic
from tianshou.utils.net.discrete import Actor, Critic
from sim.env import SimEnv
import torch
import torch.nn as nn
import time
import numpy as np
from torch.utils.tensorboard import SummaryWriter
from tianshou.utils import TensorboardLogger
from gymnasium.wrappers import FlattenObservation
HIDDEN_SIZE = 64
LAYER_NUM = 3
```

```
def ppo_run():
    """ppo algorithm."""
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    train_env = SimEnv("ppo")
    train_env = FlattenObservation(train_env)
    writer = SummaryWriter(
        'log/ppo/' + time.strftime("%Y-%m-%d-%H:%M:%S",
            time.localtime()))
    logger = TensorboardLogger(writer, 1, 1, 1, 1)
    # test_env = SimEnv()
    print(train_env.observation_space.sample())
    state_shape = train_env.observation_space.shape
    action_shape = train_env.action_space.n
    net = Net(state_shape=train_env.observation_space.shape,
              action_shape=train_env.action_space.shape,
              hidden_sizes=[64, 64, 64], device=device)
    actor = Actor(preprocess_net=net,
                  action_shape=train_env.action_space.n,
                      device=device).to(device)
    critic = Critic(preprocess_net=net, device=device).to(device)
    ac = ActorCritic(actor=actor, critic=critic)
    optim = torch.optim.Adam(ac.parameters(), lr=0.001)
    dist = torch.distributions.Categorical
    policy = ts.policy.PPOPolicy(
        actor=actor, critic=critic, optim=optim, dist_fn=dist,
            action_space=train_env.action_space,
            reward_normalization=True)
    replay_buffer = ts.data.ReplayBuffer(10000)
    train_collector = ts.data.Collector(
        policy=policy, env=train_env, buffer=replay_buffer)
    result = ts.trainer.onpolicy_trainer(
        policy=policy, train_collector=train_collector,
            test_collector=None,
        max_epoch=35, step_per_epoch=10000,
        repeat_per_collect=4, episode_per_test=1,
        batch_size=256, step_per_collect=16,
        logger=logger)
    torch.save(policy.state_dict(
    ), f"model/ppo{time.strftime('%Y-%m-%d-%H:%M:%S',
       time.localtime())}.pth")
    print(f'Finished training! Use {result["duration"]}')
def a2c_run():
    """a2c algorithm."""
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    train_env = SimEnv("a2c")
    train_env = FlattenObservation(train_env)
    writer = SummaryWriter(
        'log/a2c/' + time.strftime("%Y-%m-%d-%H:%M:%S",
            time.localtime()))
    logger = TensorboardLogger(writer, 1, 1, 1, 1)
    # test_env = SimEnv()
    print(train_env.observation_space.sample())
```

```
    state_shape = train_env.observation_space.shape
    action_shape = train_env.action_space.n
    net = Net(state_shape=train_env.observation_space.shape,
              action_shape=train_env.action_space.shape,
              hidden_sizes=[64, 64, 64], device=device)
    # net = ShareNet(train_env.observation_space.shape)
    actor = Actor(preprocess_net=net,
                  action_shape=train_env.action_space.n,
                      device=device).to(device)
    critic = Critic(preprocess_net=net, device=device).to(device)
    ac = ActorCritic(actor=actor, critic=critic)
    optim = torch.optim.Adam(ac.parameters(), lr=0.001)
    dist = torch.distributions.Categorical
    policy = ts.policy.A2CPolicy(
        actor=actor, critic=critic, optim=optim, dist_fn=dist,
           action_space=train_env.action_space)
    replay_buffer = ts.data.ReplayBuffer(1000)
    train_collector = ts.data.Collector(
        policy=policy, env=train_env, buffer=replay_buffer)
    result = ts.trainer.onpolicy_trainer(
        policy=policy, train_collector=train_collector,
           test_collector=None,
        max_epoch=30, step_per_epoch=1000,
        repeat_per_collect=4, episode_per_test=1,
        batch_size=128, step_per_collect=8,
        logger=logger)
    torch.save(policy.state_dict(
    ), f"model/a2c{time.strftime('%Y-%m-%d-%H:%M:%S',
       time.localtime())}.pth")
    print(f'Finished training! Use {result["duration"]}')
```

A.7 其他功能代码文件

```
import os
import pandas as pd

keys = ["seg_idx", "seg_size", "mAps"]
def init_tables(path):
        files = os.listdir(path)
        gt = {}
        for file in files:
            # rs, q, fr = file[:-4].split('_')
            data = pd.read_csv(f"{path}/{file}", header=None,
               names=keys)
            gt[file[:-4]] = data
        return gt

def get_gt(gt, rs, q, fr, seg_idx):
    gt_by_config = gt[f"{rs}_{q}_{fr}"]
    return gt_by_config[gt_by_config["seg_idx"] ==
       seg_idx].values.tolist()
```

```
def energy_consuming(frames_num, resolution, local=False):
    """energy consumed to process n frames.
    following the setting of paper: Joint Configuration Adaptation
       and Bandwidth Allocation for Edge-based Real-time Video
       Analytics
    @params:
        frames_num(int): frames the segment contained
        resolution(List[int]): [width, height]
        local(bool): if True the segment is processed by local
           client else by the remote server
    """
    mu = 5  # 5j/frame
    gamma = 5e-6
    alpha = 1
    width, height = resolution
    processing_energy = mu * frames_num if local else 0
    transmission_energy = alpha * gamma * \
        frames_num * (width * height * 8) ** 2 if not local else 0
    return processing_energy, transmission_energy
```

致　谢

行文至此，感慨万千，回首这段四年大学旅程，我心中充满了感激。在这里，我要衷心地感谢所有给予我帮助和支持的人们。

首先，我要由衷地感谢我的指导教师曹晨红老师。曹老师不仅在论文选题、研究方法和实验设计等方面给予了我耐心细致的指导，更在学术素养和研究思维的培养上给予了我无私的帮助。您严谨的治学态度、卓越的学术能力和对知识的深刻理解，使我受益匪浅。在曹老师的指导过程中，我不仅学到了专业知识，更收获了人生的真谛。您的悉心指导与教诲将成为我学习和成长的重要财富。

其次，我要感谢我的同学。大家四年来互相帮助，互相勉励，一起学习，一起述说愁苦，分享快乐。在这分别之际，祝大家前程似锦。

再次，我要感谢所有授课老师以及学校。感谢学校提供的良好的学习环境和发展平台，为我们的成长奠定了坚实的基础。授课老师们的辛勤付出和无私奉献，让我们能够在这里接受优秀的教育，受益终生。

最后，我要感谢我的父母。感谢你们的无私奉献和辛勤付出，让我能够专注于学业并追求自己的梦想。感谢你们对我的无条件的爱，让我坚强、自信、乐观，并始终相信自己可以克服一切困难。

感谢所有人的付出和帮助，这段旅程充满了艰辛和挑战，但也充满了成长和收获。在今后的学习和工作中，我将时刻铭记您们的教诲和关怀，不断努力提升自己，为社会做出自己的贡献。


上海大学
SHANGHAI UNIVERSITY