Tabriz Journal of Electrical Engineering (TJEE), vol. 54, no. 4, December 2024 Serial no. 110
DOI: 10.22034/tjee.2024.57664.4672

Development of Tiny Machine Learning Models for Optimal Distribution of Workloads at Edge Networks

M. Pourhosseini[1], M. Abbasi[2*], E. Mohammadi-Pasand[3]

Computer Engineering Department, Engineering Faculty, Bu-Ali Sina University, Hamedan, Iran. [1,2,3]
E-mails: m.pourhoseini@eng.basu.ac.ir; abbasi@basu.ac.ir; e.mohammadipasand01@eng.basu.ac.ir

Short Abstract
The number of devices connected to the Internet of Things has been expanded rapidly. This issue has caused a significant increase in the computational load in the networks. To overcome this challenge, cloud computing was presented as a suitable solution. However, cloud computing suffers significant delay to process workloads. Processing workloads at the edge of the network and locally leads to reduced latency. But due to the limitation of computing resources at the edge, managing and optimizing resources is considered one of the main challenges. Therefore, in addition to distributing the workloads at the edge of the network and maintaining the balance between energy consumption and delay, the limitation of resources such as memory consumption should be considered. In this paper, an online method based on XCS learning classifier systems (LCS), named TinyXCS, and an offline method based on decision tree, named TinyDT, are proposed to balance energy consumption and reduce delay in processing workloads considering the memory limitation at edge. The experimental results show the superiority of TinyXCS and TinyDT over similar methods. The simulation shows that in addition to workload distribution, the proposed methods can simultaneously reduce delay and energy consumption and create a compromise between them and memory consumption.

Keywords
Internet of things, edge computing, machine learning, workload distribution, quality of service.

1- Short Introduction

As the Internet of Things continues to expand and the number of connected devices increases, the volume of information generated by these devices is also growing. This poses challenges for network. To address this issue, cloud-fog-edge computing is employed to effectively manage and store this vast amount of information. In this article, we consider the quality of service and the benefits of edge computing, such as the close proximity of computing resources to objects and minimal latency. We propose the utilization of machine learning models to achieve balanced workload allocation at the network's edge. But due to the limitation of computing resources at the edge of the network, we have to use tiny machine learning models.

2- Proposed Work and Methodology

In this work, two methods are proposed for workload allocation according to the quality of service and the limitation of computing resources in edge devices. The first method uses an XCS algorithm with limited memory consumption called TinyXCS as a method that does not know the future of the system (online) to allocate workloads at the edge of the network. The main goal of XCS is to create a complete and accurate map of the problem space. In the XCS algorithm, a learning problem is defined as an action selection problem that the agent must choose an action to solve. Using a reinforcement learning algorithm, the agent tries to select the best action from a set of actions for the problem, and then the rules for this action are updated with the aim of increasing accuracy. In this research, the algorithm is employed to make informed decisions in determining the manageable workload at the network's edge. According to the modelling, the input of the problem, which is also known as the network status, includes the input load, network congestion, and the battery status, which is determined according to green energy. Furthermore, the problem's output, which is the load that can be processed at the edge, is implemented as an action within the environment. The second method is a tiny machine learning model based on a decision tree called TinyDT. In this model, it is known as an offline system due to the model's knowledge of the future situation. In this model, the initial step involves training a decision tree model using the input dataset. This dataset comprises input load parameters, network congestion, energy, and processing load. Subsequently, the system utilizes this trained model to make decisions. Both methods take into account the constraint of limited computing resources on edge devices and provide the ability to manage the memory consumption in these systems. The results obtained from the TinyXCS model demonstrate that, even with a 50% reduction in memory compared to similar models, the delay cost converges to 3 milliseconds. Also, the offline model of TinyDT with a maximum memory of 3 KB has a maximum delay of 4 milliseconds

3- Conclusion

We introduced a method to optimize the distribution of workloads and balance the latency and consumption of energy and computing resources at the edge. In the proposed method, first an LCS called TinyXCS and then a reduced decision tree model called TinyDT were used. The proposed methods showed that in addition to improving energy consumption and delay, they are compatible with the limitation of computing resources at the edge of the network.

4- References

[1] X. Niu et al., "Workload allocation mechanism for minimum service delay in edge computing-based power internet of things," IEEE Access, vol. 7, pp. 83771-83784, 2019.
[2] M. Abbasi, M. Yaghoobikia, M. Rafiee, A. Jolfaei, and M. R. Khosravi, "Efficient resource management and workload allocation in fog–cloud computing paradigm in IoT using learning classifier systems," Computer Communications, vol. 153, pp. 217-228, 2020.

توسعه مدل‌های کوچک یادگیری ماشین برای توزیع بهینه بارهای کاری در شبکه‌های لبه

محمدرضا پورحسینی
دانشجوی کارشناسی ارشد معماری سیستم‌های کامپیوتری، گروه مهندسی کامپیوتر، دانشکده مهندسی، دانشگاه بوعلی سینا، همدان، ایران

مهدی عباسی
دانشیار، گروه مهندسی کامپیوتر، دانشکده مهندسی، دانشگاه بوعلی سینا، همدان، ایران

احسان محمدی‌پسند
کارشناسی ارشد شبکه‌های کامپیوتری، گروه مهندسی کامپیوتر، دانشکده مهندسی، دانشگاه بوعلی سینا، همدان، ایران

چکیده
تعداد دستگاه‌های متصل به شبکه اینترنت اشیاء به سرعت گسترش یافته است. این امر، موجب افزایش قابل توجه بار محاسباتی در شبکه‌ها شده است. برای غلبه بر این چالش، محاسبات ابری به عنوان یک راهکار مناسب ارائه شد. اما محاسبات ابری تأخیر قابل توجهی را برای پردازش بارها متحمل می‌شد. پردازش بارهای کاری در لبه شبکه و به صورت محلی تاخیر را کاهش می‌دهد. اما، به دلیل محدودیت منابع محاسباتی در لبه شبکه، مدیریت منابع و بهینه‌سازی استفاده از آنها از چالش‌های اساسی در پردازش لبه است. بنابراین علاوه بر توزیع بارکاری در لبه شبکه و حفظ تعادل بین انرژی مصرفی و تأخیر، باید محدودیت منابع مانند حافظه مصرفی را در نظر گرفت. در این مقاله، یک روش برخط مبتنی بر سیستم‌های طبقه بندی کننده یادگیری XCS (LCS)، با نام TinyXCS و یک روش برون‌خط مبتنی بر درخت تصمیم با نام TinyDT، برای متعادل کردن مصرف انرژی و کاهش تأخیر در پردازش بارهای کاری با درنظر گرفتن محدودیت حافظه در لبه شبکه پیشنهاد شده است. نتایج آزمایش‌های ما نشان‌دهنده برتری TinyXCS و TinyDT نسبت به روش‌های مشابه است. شبیه‌سازی نشان می‌دهد که روش‌های پیشنهادی می‌توانند علاوه بر توزیع بارکاری، سبب کاهش همزمان تأخیر و انرژی مصرفی و ایجاد مصالحه بین آن‌ها و حافظه مصرفی شوند.

کلمات کلیدی
اینترنت اشیاء، محاسبات لبه، یادگیری ماشین، توزیع بارکاری، کیفیت سرویس.

نام نویسنده مسئول: مهدی عباسی
ایمیل نویسنده مسئول: abbasi@basu.ac.ir

تاریخ ارسال مقاله: ۱۴۰۲/۰۵/۲۹
تاریخ(های) اصلاح مقاله: ۱۴۰۲/۰۹/۲۳
تاریخ پذیرش مقاله: ۱۴۰۲/۱۱/۳۰

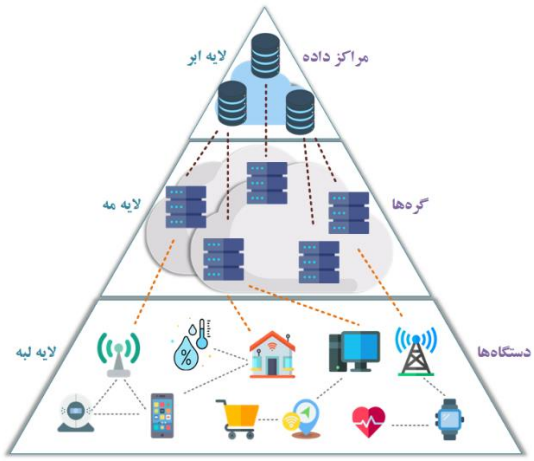
۱- مقدمه

اینترنت اشیاء[1] به مجموعه‌ای از دستگاه‌ها و اشیاء فیزیکی گفته می‌شود که از طریق اینترنت، اطلاعات را با یکدیگر تبادل می‌کنند[۶]. اطلاعات تولید شده در اینترنت اشیاء، با هدف کنترل و نظارت بر اشیاء، اتخاذ تصمیمات و بهبود عملکرد به سرویس‌دهنده‌ها یا سیستم‌های مرکزی ارسال می‌شود. از مزایای اینترنت اشیاء می‌توان به افزایش کارایی، بهبود ارتباطات و کاهش هزینه‌ها اشاره کرد[۷].

با ورود اینترنت اشیاء به زندگی روزمره و توسعه آن در بسیاری از حوزه‌ها، کیفیت زندگی بهبود یافته است. این فناوری در خانه، محل کار، سفر و بیمارستان به کار گرفته می‌شود[۸]. استفاده از اینترنت اشیاء با چالش‌های مهمی همراه شده است. از جمله این چالش‌ها میتوان به امنیت و حریم خصوصی، مدیریت داده، سازگاری، پایداری و مقیاس‌پذیری اشاره کرد[۶]. برای غلبه‌بر چالش‌های اینترنت اشیا، محاسبات ابر[2] به عنوان یک راهکار ارائه شده است[۹]. در محاسبات ابر توابع محاسباتی، ذخیره‌سازی و مدیریت شبکه به سمت مراکز داده‌ی متمرکز منتقل می‌شوند. در این مدل، منابع محاسباتی مرکزی در قالب زیرساخت‌های ابری فراهم می‌شود[۱۰]. با وجود بهبود چالش‌های اینترنت اشیاء، استفاده از محاسبات ابر چالش‌هایی مانند امنیت و حریم خصوصی، عملکرد و کارایی، قابلیت پیش‌بینی و مقیاس‌پذیری و قابلیت انتقال داده را به همراه داشته است[۱۱]. برای پرداختن به این چالش‌ها شرکت سیسکو در سال ۲۰۱۴ مفهوم محاسبات مه[3] را ارائه داد. در این مدل منابع محاسباتی و ذخیره‌سازی به کاربران نهایی نزدیک می‌شود. هدف از محاسبات مه پردازش محلی بخشی از حجم کار در دستگاه‌های مه است. در این مدل، منابع محاسباتی روی دستگاه‌های محلی مانند مسیریاب‌ها، سوییچ‌ها، و حسگرها قرار می‌گیرند و امکان پردازش و تحلیل داده‌ها را به صورت محلی فراهم می‌کند[۱۲]. با وجود این مزایا، محاسبات مه برای انجام کارهای حساس به زمان مناسب نبود و در پاسخ به این چالش، محاسبات لبه[4] در سال ۲۰۱۷ معرفی شد. بزرگترین مزیت محاسبات لبه نسبت به محاسبات مه کاهش تأخیر به دلیل پردازش بار کاری در نزدیکی اشیاء است. محاسبات لبه یک مدل محاسباتی است که در آن، منابع محاسباتی و ذخیره‌سازی داده‌ها تا حد امکان به منابع محلی ودستگاه‌های اینترنت اشیاء نزدیک می‌شود[۱۳]. شکل ۱ معماری سه لایه ابر-مه-لبه را نشان می‌دهد. همانطور که در شکل ۱ نشان داده شده است، معماری ابر-مه-لبه از سه لایه با ظرفیت محاسباتی متفاوت تشکیل

[1] Internet of things(IOT)
[2] Cloud computing
[3] Fog computing
[4] Edge computing

می‌شود این لایه‌ها عبارتنداز: لایه ابر: در بالاترین سطح از این معماری لایه ابر قرار دارد. در این لایه مراکز داده قدرتمند با ظرفیت محاسباتی بالایی با قابلیت پردازش و ذخیره سازی داده‌های حجیم وجود دارند. به علت فاصله بین اشیاء و دستگاه‌های پردازشی تأخیر پردازش اطلاعات زیاد است. لایه مه: این لایه بین لایه ابر و لبه قرار دارد و دستگاه‌های پردازشی و ذخیره‌سازی را به اشیاء نزدیکتر می‌کند. از ویژگی‌های این لایه کاهش تأخیر پردازشی داده‌ها است. لایه لبه: لایه لبه به عنوان بخشی از لایه مه، دستگاه‌های پردازشی و ذخیره‌سازی را در لبه شبکه و نزدیک به اشیاء و کاربران نهایی قرار می‌دهد. در لایه لبه امکان پردازش داده‌های محدود اما با کارایی بالا و تأخیر زمان واقعی[5] فراهم می‌شود. در صورت عدم توانایی منابع محاسباتی در لبه شبکه، این بارها به سمت لایه بالاتر فرستاده می‌شوند. در لایه مه، پردازش به صورت نیمه متمرکز توسط مسیریاب‌ها انجام می‌شود و امکان انجام عملیات محاسباتی با تأخیر کم بدون ارسال به ابر را فراهم می‌کند. در صورت ازدحام در شبکه و عدم توانایی پردازش بارها در مه، بارکاری برای پردازش به لایه بالاتر ارسال می‌شود. در لایه ابر، داده‌ها به یک یا چند مرکز داده بزرگ و مرکزی ارسال می‌شوند. در این مراکز داده، پردازش و ذخیره‌سازی داده‌ها به طور متمرکز انجام می‌شود. با حرکت به سمت لایه‌های بالاتر قدرت پردازشی و تأخیر افزایش می‌یابد.



شکل ۱- اینترنت اشیاء و معماری سه لایه ابر-مه-لبه[۱۴]

با وجود کاهش تأخیر، بهبود عملکرد و کارایی، حفظ حریم خصوصی و کاهش بار مرتبط با ارتباطات در محاسبات لبه، با چالش‌های اساسی ازجمله توان محاسباتی محدود، مدیریت و توزیع منابع، مدیریت داده روبرو هستیم[۱۴].

به دلیل محدود بودن منابع محاسباتی دستگاه‌های لبه، با چالش توزیع بارکاری بین این منابع رو برو هستیم. از این رو تصمیم‌گیری در مورد میزان بارکاری ارسالی به دستگاه‌ها باید در نظر گرفته شود و همچین در صورت اتمام منابع و یا وجود ازدحام در شبکه، چه میزان از بارکاری برای پردازش به لایه‌های بالاتر ارسال شود.

توزیع متعادل بارکاری در لبه شبکه نیازمند استفاده از الگوریتم‌ها، روش‌ها و معماری‌های مناسب است که با در نظر گرفتن ویژگی‌ها و چالش‌های بالا، بارکاری را به صورت متعادل و بهینه بین دستگاه‌های لبه توزیع شود[۱۵]. برای این منظور پژوهش‌هایی برای توزیع بار و کاهش مصرف انرژی و تأخیر و تعادل بین آنها انجام شده‌است. در این پژوهش‌ها از روش‌هایی مانند الگوریتم‌های خوشه‌بندی، یادگیری تقویتی و الگوریتم‌های اکتشافی استفاده شده‌است[۱۴]. با توجه به اینکه این الگوریتم‌ها اغلب به حافظه و CPU بالایی نیاز دارند و پیاده‌سازی و نگهداری این الگوریتم‌ها توسط دستگاه‌های لبه با منابع محاسباتی محدود پرهزینه و دشوار است. بنابراین، در این مقاله درتلاشیم راه حلی با استفاده از یادگیری ماشین و توجه به کیفیت سرویس و مدیریت منابع ارائه دهیم. اخیراً یادگیری ماشین به عنوان یک راه حل هوشمند برای پیش‌بینی وضعیت آینده و تصمیم‌گیری برای دستیابی به بهترین نتیجه در زمینه‌های مختلف مورد استفاده قرار گرفته شده‌است[۱۶, ۱۷]. استفاده از یادگیری ماشین برای توزیع بار کاری در لبه شبکه و تصمیم‌گیری برای رسیدن به توزیع متعادل یا ارسال داده‌های پردازشی به لایه‌های بالاتر برای مدیریت منابع محاسباتی بسیار کارآمد است. اما به دلیل محدودیت منابع محاسباتی مانند حافظه در لبه شبکه، ضرورت استفاده از مدل‌های سبک وزن یادگیری ماشین بیش از پیش احساس می‌شود. این مدل‌ها به منابع محاسباتی کمتری نسبت به مدل‌های پایه نیاز دارند[۱۵].

در روش پیشنهادی، علاوه برکیفیت سرویس به محدودیت منابع محاسباتی در دستگاه‌های لبه برای توزیع بارکاری نیز توجه خواهیم کرد. برای این منظور یک مدل یادگیری ماشین کوچک بر اساس درخت تصمیم که اطلاعات کافی در مورد آینده سیستم دارد (به صورت برون‌خط) ارائه شده است. همچنین الگوریتم XCS با مصرف حافظه محدود را به عنوان روشی که از آینده سیستم اطلاعی ندارد (به صورت برخط) برای توزیع بارهای کاری در لبه شبکه استفاده می‌شود. ساده بودن ساختار داده لازم برای دو مدل برخط و برون خط مبتنی بر XCS و درخت تصمیم امکان پیاده‌سازی آنها را در تراشه‌های برنامه‌پذیر و حافظه های تداعیگر فراهم می‌کند. در اینصورت، می‌توان از فناوری پردازش درون شبکه‌ای که از طریق پردازش مدل‌های کوچک در سوییچ‌های برنامه پذیر و حافظه‌های تداعیگر فراهم می‌شود، برای تسریع بیشتر این مدل‌ها استفاده کرد.

مشارکت‌های این مقاله به شرح زیر است:

(الف) ارائه دو مدل هوشمند آنلاین و آفلاین برای تخصیص حجم کار در لبه شبکه،

(ب) مدیریت مصرف حافظه در مدلها ی پیشنهادی با توجه به محدودیت منابع محاسباتی در لبه شبکه،

(ج) توزیع متعادل بارهای کاری و حفظ مصالحه بین مصرف انرژی و تاخیر پردازش حجم کاری و تاخیر ارتباط بین دستگاه‌های لبه.

۲- مروری بر کارهای پیشین

جدول شماره ۱ مهمترین تحقیات انجام شده را در زمینه توزیع بارهای کاری در شبکه‌های لبه نشان می‌دهد. در ادامه، به توضیح و بررسی دقیق این کارها می‌پردازیم.

Li و همکارانش[۱۸] در سال ۲۰۱۸ الگوریتم GLOBE[6] را معرفی کردند. آن‌ها در این الگوریتم تعادل بین بار جغرافیایی و کنترل ورودی بارها را برای بهینه‌سازی عملکرد ایستگاه‌های لبه شبکه با هم ترکیب کردند. این الگوریتم به صورت برخط و بدون اطلاع از اطلاعات آینده سیستم به چالش‌های مهمی ناشی از وضعیت باتری و محدودیت‌های انرژی پاسخ می‌دهد. در این پژوهش معیارهای تأخیر محاسباتی بارکاری و انرژی مصرفی برای ترافیک داده‌ها و ارتباط بین ایستگاه‌های لبه در نظر گرفته شده است. همچنین از ترافیک داده‌های غیر قابل پردازش توسط انرژی تجدیدپذیر و تأخیر انتقال بی‌سیم به هوا صرف نظر شده است. الگوریتم GLOBE در مقایسه با الگوریتم‌های برون‌خط هزینه عملیاتی سیستم را تا ۵۰٪ کاهش می‌دهد و بین ظرفیت باتری و عملکرد سیستم تعادل نسبی ایجاد می‌کند. الگوریتم ذکرشده با وجود اینکه توزیع

[5] Real time

[6] Geographical Load Balancing Edge

بارهای محاسباتی در گره‌های شبکه را بهبود بخشید اما برای اجرا به منابع محاسباتی بالایی نیاز دارد.

Wan و همکارانش[۱۹] در سال ۲۰۱۸ یک روش متعادل‌سازی و زمان‌بندی بار برای بهبود مصرف انرژی مبتنی بر محاسبات مه را پیشنهاد دادند. آن‌ها ابتدا مدل مصرف انرژی را برای تعادل بارهای کاری روی گره‌های مه ایجاد کردند. سپس با الگوریتم بهینه‌سازی دسته‌ای این متعادل‌سازی را بهبود بخشیدند. با آزمایش روش پیشنهادی بر روی ربات‌ها و raspberry pi نشان داده شد که بارکاری به‌صورت متوازن در میان ربات‌ها توزیع شده است. با وجود بهبودکارایی و مصرف انرژی، تعادل بین مصرف انرژی و تأخیر نادیده گرفته شده بود.

Niu و همکارانش[۲] در سال ۲۰۱۹ یک سازوکار تخصیص بارکاری مبتنی بر محاسبات لبه به نام BRT را برای به حداقل رساندن تأخیر سرویس معرفی کردند. این سازوکار شامل سه بخش اصلی می‌شود، در بخش اول، الگوریتم متعادل‌سازی تخصیص وظایف برای بهبود تعادل بارکاری در میان گره‌های لبه شبکه ایجاد می شود. این الگوریتم دو معیار فاصله بین ترمینال و گره لبه و همچنین قابلیت محاسباتی گره لبه برای پردازش بارکاری اختصاص داده شده را در نظر می‌گیرد. در بخش دوم، الگوریتم بهینه‌سازی ازدحام ذرات(MPSO[۷]) برای بهبود تخصیص منابع CPU گره‌های لبه بررسی شد و در بخش سوم یک الگوریتم برنامه نویسی نیمه معین برای تخصیص وظایف به گره‌های لبه به کار گرفته شد. نتایج شبیه‌سازی نشان می‌دهد که الگوریتم پیشنهادی بهتر از الگوریتم‌های SSA[۸]، LAB[۹] و LEAD[۱۰] در حل چالش به حداقل رساندن تأخیر سرویس عمل می‌کند. اما الگوریتم MPSO استفاده شده در بخش تخصیص منابع به دلیل پیچیدگی، توان مصرفی بالایی نیاز دارد. از دیگر ضعف‌های این سازوکار نیاز به منابع مصرفی بالا برای اجرای الگوریتم‌ها در هر بخش است.

Abbasi و همکارانش[۳] در سال ۲۰۲۰ روشی بر پایه سیستم‌های طبقه‌بندی یادگیری به نام BCM-XCS برای متعادل‌کردن مصرف انرژی در لبه شبکه و کاهش تأخیر در پردازش بارهای کاری پیشنهاد دادند. در مدل‌سازی سیستم، تأخیر کل سیستم شامل سه تأخیر انتقال بارکاری به شبکه بی سیم، تأخیر پردازش محلی بارهای کاری در لبه و تأخیر انتقال بارکاری به ابر است. همچنین انرژی مصرفی کل سیستم به دو بخش توان عملیاتی و توان محاسباتی تقسیم می‌شود. در این پژوهش در صورتی که ظرفیت باتری کمتر از انرژی مورد نیاز برای پردازش بارهای کاری در لبه باشد، همه بار به سمت ابر ارسال می‌شود در غیر این صورت پردازش بارکاری در لبه انجام می‌شود. نتایج حاصل از پیاده‌سازی این الگوریتم نشان داد که تأخیر کلی سیستم تا ۴۲٪ کاهش یافت همچنین در صورت استفاده از باتری‌هایی با منبع انرژی تجدیدپذیر، شاهد بهبود مصرف انرژی هستیم. از ضعف‌های این سیستم می‌توان به مصرف بالای منابع محاسباتی آن اشاره کرد.

Abbasi و همکارانش[۴] در سال ۲۰۲۱ چارچوبی را با استفاده از الگوریتم NSGA II برای ایجاد مصالحه بین مصرف انرژی و تأخیر دستگاه‌های مه در لبه شبکه ارائه دادند. درمقاله مذکور، توزیع بارکاری بر اساس دو معیار تأخیر و انرژی مصرفی انجام می‌شود. تأخیر کلی شامل تأخیر در پردازش درخواست‌ها در گره ترمینال، تأخیر پردازش درخواست‌ها در دستگاه‌های مه و تأخیر انتقال بسته‌ها در شبکه بی‌سیم بین گره پایانه و کنترل‌کننده مرکزی است. همچنین انرژی مصرف شده برای پردازش بسته‌ها در دستگاه مه شامل انرژی مصرف شده برای پردازش بسته‌ها در دستگاه مه و انرژی مصرفی در هر سیکل CPU است. با توجه به نتایج شبیه‌سازی می‌توان مشاهده کرد که الگوریتم NSGA II نسبت به الگوریتم‌های پیشین تأخیر و مصرف انرژی را کاهش می‌دهد. از ضعف‌های این الگوریتم می‌توان به حافظه مصرفی بالا و سرعت پایین آن اشاره کرد.

جدول ۱- مهمترین کارهای انجام شده در زمینه توزیع بهینه بارهای کاری در شبکه‌های لبه

مقاله	هدف	روش	مزایا	معایب
Li و همکاران[۱]	تخصیص بارکاری مبتنی بر تئوری صف با هدف تعادل بین مصرف انرژی و تأخیر	استفاده از از برنامه‌ریزی غیرخطی برای کاهش مصرف انرژی و الگوریتم STML برای کاهش تأخیر	مصرف انرژی تا ۲۲ درصد و تأخیر تا ۱۲.۵ درصد کاهش یافت	نیاز به منابع محاسباتی سنگین جهت پردازش الگوریتم
Niu و همکاران[۲]	تخصیص بارکاری مبتنی بر محاسبات لبه برای به حداقل رساندن تأخیر سرویس معرفی کردند	الگوریتم BRT با هدف مقدار دهی اولیه متعادل، تخصیص منابع و تخصیص وظیفه ارائه شد	موفقیت در به حداقل رساندن تأخیر نسبت به الگوریتم‌های مشابه	توان مصرفی بالا و عدم مصالحه بین انرژی، تأخیر و منابع محاسباتی
Abbasi و همکاران[۳]	متعادل‌کردن مصرف انرژی در لبه شبکه و کاهش تأخیر در پردازش بارهای کاری	استفاده از الگوریتمی بر پایه قوانین و حافظه‌دار BCM	کاهش ۴۲ درصدی تأخیر کلی سیستم و بهبود مصرف انرژی	در نظر نگرفتن حافظه مصرفی الگوریتم BCM
Abbasi و همکاران[۴]	ایجاد مصالحه بین مصرف انرژی و تأخیر دستگاه‌های مه در لبه شبکه	استفاده از الگوریتم NSGA II برای توزیع بارکاری در لبه شبکه	کاهش نسبی تأخیر و انرژی مصرفی	سرعت پایین الگوریتم و در نظر نگرفتن حافظه مصرفی
Abbasi و همکاران[۵]	توزیع بارهای کاری در یک سیستم محاسباتی لبه-ابر و اینترنت اشیاء	به حداقل رساندن مصرف انرژی و تأخیر و کاهش مصرف انرژی شبکه خودرو	کاهش ۴۰ درصدی تأخیر و کاهش مصرف باتری با استفاده از انرژی‌های تجدیدپذیر	کاهش کارایی کارگزاران لبه در ارضا نیازهای پردازش بلادرنگ با کاهش میزان انرژی نو

[7] Modified particle swarm optimization
[8] Simulated annealing algorithm
[9] LoAd Balancing
[10] Latency-awarE workloAd offloaDing

۳- روش پیشنهادی

با توجه به بخش قبل در روش‌های بیان شده مشکلاتی مانند مصرف بالای منابع محاسباتی از جمله حافظه مصرفی وجود داشت که ما را بر آن داشت تا با ارائه روشی کارآمد، توزیع بارکاری در لبه شبکه را بهبود دهیم. از آنجا که برخی از جریان‌های کاری حساسیت زیادی به تأخیر در زمان پاسخ دارند، نمی‌توان آن‌ها را به سمت سیستم‌های ابر فرستاد و به ناچار باید این محاسبات در گره‌های لبه شبکه صورت پذیرد؛ همچنین، سیستم‌های مبتنی بر یادگیری ماشین و تنظیم بارکاری در لبه و مه سیستم‌های سنگینی هستند ولی در گره‌های لبه منابعی مانند توان محاسباتی و حافظه محدود است و اجرای الگوریتم یا برنامه‌های با هدف زمانبندی کارها و یا تخصیص جریان کاری با حجم حافظه بالا در حافظه ماشین لبه نمی‌گنجد. پس باید از سیستم‌های سبک‌تری از یادگیری ماشین برای زمانبندی بارکاری بر اساس منابع موجود در پردازش لبه استفاده کرد. در این مقاله به دنبال راه حلی کارآمد بر مبنای مدل‌های کوچک یادگیری ماشین برای زمانبندی بارکاری براساس منابع پردازشی در لبه‌ی شبکه است.

ایده اصلی در این پژوهش استفاده از مدل‌های یادگیری ماشین کوچک اما کارآمد برای توزیع بارکاری در لبه شبکه است که با در نظر گرفتن پارامترهایی مانند بار ورودی شبکه، ازدحام شبکه، انرژی مصرفی برای پردازش بارهای کاری و همچنین تأخیر کلی سیستم، تصمیم بگیرد که کدام منبع برای چه مدت زمانی برای چند درصد از کار تخصیص داده شود. در ادامه پس از ارائه مدل جامع سیستم لبه، به بررسی روش‌های پیشنهادی می‌پردازیم.

۳-۱- مدل‌سازی سیستم

مدل و سناریوهای در نظر گرفته شده در این مقاله بر اساس مدل‌ها و سناریوهای ارائه شده در مقاله‌های پیشین نویسندگان مقاله است. بر اساس این مدل و سناریوها می‌توان نشان داد که مدلهای کوچک یادگیری ماشین توانایی به حداقل رساندن هزینه کلی سیستم را در توزیع متعادل بارهای کاری در لبه شبکه دارند. البته، هدف اصلی آن است که اندازه مدل یادگیری ماشین مورد استفاده برای تصمیم‌گیری در مورد نحوه توزیع بارهای کاری را تا جایی‌که منجر به افت چشمگیر در کارایی آن نشود کاهش داد.

ما یک سیستم لبه شامل یک ایستگاه پایه و مجموعه‌ای از سرویس‌دهنده‌های لبه‌ای که به صورت فیزیکی در کنار هم قرار گرفته‌اند را درنظر می‌گیریم. ایستگاه پایه مسئولیت تصمیم‌گیری در مورد میزان بارکاری قابل پردازش در لبه و همچنین ارسال بار به لایه‌های بالاتر را بر عهده دارد. هر منبع پردازشی در لبه شبکه دارای یک منبع انرژی با ظرفیت محدود است بنابراین، از یک منبع انرژی مشترک در شبکه برای شارژ باتری‌های منابع پردازشی استفاده می‌شود. جدول ۱ پارامترهای مدل‌سازی سیستم را به اختصار نشان می‌دهد.

مدل بارکاری: در این سناریو مدل زمانی به صورت گسسته و $t = 0,1,2,\dots$ فرض می‌شود. $\alpha(t)$ برابر بارکاری ورودی در لحظه t به لبه شبکه را بیان می‌کند. سیستم تصمیم می‌گیرد چه مقدار از این بار ورودی را به صورت محلی در لبه شبکه پردازش شود که این مقدار با پارامتر $\beta(t)$ نشان داده می‌شود. در نهایت مقدار بارکاری $\alpha(t) - \beta(t)$ به لایه بالاتر ارسال می‌شود. همچنین تعداد سرویس‌دهنده‌های فعال در هر بازه زمانی برابر است $n(t) \in [0, N]$ که N برابر حداکثر تعداد سرویس‌دهنده‌ها در لبه شبکه است.

جدول ۲- پارامترهای مدل‌سازی

معنا	نماد	معنا	نماد
وضعیت سیستم	$S(t)$	بار ورودی	$\alpha(t)$
هزینه تأخیر پردازشی در لبه شبکه	$d_{pro}(t)$	بارکاری پردازش شده در لبه	$\beta(t)$
هزینه تأخیر ارسال بار به لایه بالاتر	$d_{tra}(t)$	هزینه استفاده از منبع تغذیه پشتیبان	$c_{back}(t)$
هزینه کل تأخیر	$D_{sum}(t)$	میزان انرژی تجدیدپذیر	$E_{gr}(t)$
تعداد سرویس‌دهنده‌های فعال	$n(t)$	ازدحام شبکه	$C(t)$
انرژی مصرفی عملیاتی	$e_{op}(t)$	کل انرژی مصرفی	$E_{sum}(t)$
انرژی مصرفی پردازشی	$e_{pro}(t)$	سطح باتری در لبه	$B(t)$

مدل تأخیر: در این سیستم دو دسته تأخیر وجود دارد.

الف) تأخیر پردازش بارهای کاری در لبه شبکه: این تأخیر به پارامترهای $n(t)$ و $\beta(t)$ یعنی تعداد سرویس‌دهنده‌های فعال و میزان بارکاری قابل پردازش در لبه شبکه وابسته است. محاسبه این تأخیر با توجه سازوکار مدیریت صف در سرویس‌دهنده‌های لبه انجام می‌شود. درنتیجه تأخیر پردازش بارهای کاری در لبه شبکه از رابطه (۱) محاسبه می‌شود که در این رابطه c ظرفیت پردازشی هر سرویس‌دهنده برحسب تعداد درخواست‌ها در ثانیه است.

$$d_{pro}(t) = \frac{\beta(t)}{n(t).c - \beta(t)} \quad (1)$$

ب) تأخیر ارسال باقیمانده بارهای کاری به لایه بالاتر: این تأخیر که با $d_{tra}(t)$ نمایش داده می‌شود به وضعیت ازدحام در شبکه وابسته است. ازدحام در شبکه که با نماد C(t) نشان داده می‌شود که به معنی ترافیک در شبکه و تأخیر در ارسال و دریافت اطلاعات در لایه مه است. درنتیجه تأخیر ارسال باقیمانده بارهای کاری به لایه بالاتر که به C(t) وابسته است از رابطه (۲) به‌دست می‌آید.

$$d_{tra}(t) = (\alpha(t) - \beta(t))C(t) \quad (۲)$$

در نهایت تأخیر کل بارهای ورودی به شبکه که با نماد $D_{sum}(t)$ مشخص می‌شود از مجموع دو تأخیر بالا به‌دست می‌آید در رابطه (۳) نشان داده شده است.

$$D_{sum}(t) = d_{pro}(t) + d_{tra}(t) \quad (۳)$$

مدل مصرف انرژی: در این سیستم مدل مصرف انرژی نیز مانند مدل تأخیر به دو دسته تقسیم می‌شود.

الف) انرژی لازم برای عملیات پایه و انتقال بارها در لبه شبکه: این انرژی فقط به میزان بارکاری ورودی به لبه شبکه بستگی دارد و به دو پارامتر انرژی مصرفی ثابت در لبه شبکه و انرژی مصرفی متغیر بر اساس بار ورودی در لبه شبکه وابسته است که به ترتیب با نمادهای e_{fixed} و $e_{dyn}(\alpha(t))$ نشان داده می‌شود. در نتیجه انرژی لازم برای عملیات پایه و انتقال بارها در لبه شبکه که از جمع این دو پارامتر حاصل می‌شود با نماد $e_{op}(t)$ مشخص و در رابطه (۴) نمایش داده شده است.

$$e_{op}(t) = e_{fixed} + e_{dyn}(\alpha(t)) \quad (۴)$$

ب) انرژی لازم برای پردازش بارهای کاری در لبه شبکه: این انرژی که با نماد $e_{pro}(t)$ نشان داده می‌شود براساس میزان بار قابل پردازش در لبه شبکه $\beta(t)$ و تعداد سرویس‌دهنده‌های فعال n(t) محاسبه می‌شود. رابطه (۵) نحوه محاسبه این انرژی را بیان می‌کند.

$$e_{pro}(t) = n(t) * \beta(t) \quad (۵)$$

در نهایت انرژی مصرفی کل که با نماد $E_{sum}(t)$ مشخص می‌شود از مجموع دو انرژی بالا به‌دست می‌آید در رابطه (۶) نشان داده شده است.

$$E_{sum}(t) = e_{pro}(t) + e_{op}(t) \quad (۶)$$

مدل باتری: در این سناریو برای هر سرویس‌دهنده لبه یک باتری با ظرفیت محدود درنظر گرفته‌شده که با نماد B(t) مشخص شده‌است. این باتری‌ها توسط انرژی خورشیدی و بادی شارژ می‌شوند. میزان بارکاری پردازشی در لبه شبکه به سطح باتری سرویس‌دهنده‌ها وابسته است در نتیجه دو وضعیت برای باتری‌ها در نظر گرفته می‌شود.

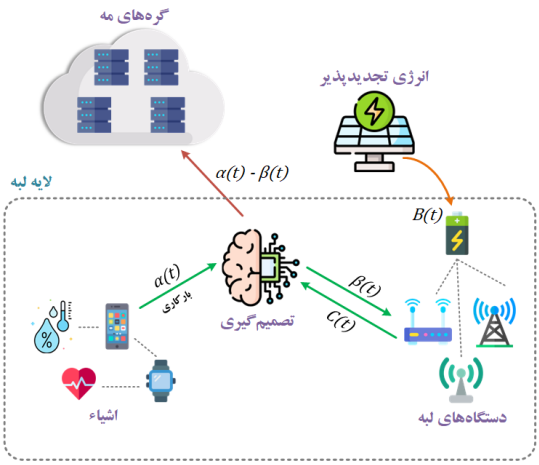
الف) هنگامی که ظرفیت باتری کمتر از انرژی لازم برای انجام عملیات پایه و انتقال بارها در لبه شبکه است. در این حالت طبق رابطه (۷) باتری‌ها برای پردازش در بازه زمانی بعدی توسط انرژی تجدیدپذیر شارژ می‌شوند.

$$B(t+1) = B(t) + E_{gr}(t) \quad (۷)$$

ب) اگر سطح باتری بیشتر از انرژی مصرفی لازم برای انجام عملیات پایه باشد، بخشی از بارکاری در لبه شبکه پردازش شده و باقیمانده آن به لاه بالاتر ارسال می‌شود در نتیجه طبق رابطه (۸) سطح باتری برای بازه بعدی محاسبه می‌شود.

$$B(t+1) = B(t) + E_{gr}(t) - E_{sum}(t) \quad (۸)$$

با توجه به چهار مدل بیان شده، معماری سیستم پیشنهادی در شکل ۲ نشان داده شده است. در این شکل $\alpha(t)$ به عنوان بارکاری ورودی از سمت اشیاء و کاربران نهایی وارد سیستم تصمیم‌گیری مبتنی بر یادگیری ماشین می‌شود. در این مقاله ما دو مدل سیستم تصمیم‌گیری برون‌خط و برخط را معرفی می‌کنیم. این سیستم تصمیم می‌گیرد چه میزان از بار برای پردازش محلی به دستگاه‌های لبه($\beta(t)$) و چه میزان به لایه بالاتر ارسال شود($\alpha(t) - \beta(t)$). در بخش‌های بعد دو مدل TinyDT و TinyXCS به صورت مدل‌های برون‌خط و برخط، به ترتیب معرفی می‌شوند.



شکل ۲- معماری سیستم پیشنهادی بر پایه چهار مدل بیان شده

۲-۳- سیستم دسته‌بند یادگیر

سیستم دسته‌بند یادگیر(LCS[11]) یک عامل هوشمند است که با محیط در تعامل است. این الگوریتم با توجه به شرایط فعلی محیط بهترین عمل را انجام می‌دهد و با گذشت زمان و افزایش دفعات فرآیند یادگیری بهبود می‌یابد[۲۰]. فرآیند یادگیری این الگوریتم به نحوی پیاده‌سازی می‌شود که با استفاده از روش‌های جستجوی شهودی و الگوریتم‌های تکاملی به دنبال جستجوی قوانین بهتر است. نسخه‌های ابتدایی این الگوریتم در ابتدا برای توسعه الگوریتم‌های ژنتیک ارائه شد که به دلیل پیچیدگی زیاد پیاده‌سازی آن دشوار بود[۲۱]. سال‌ها بعد نوعی از LCS با نام XCS ارائه شد که در آن برازش قوانین بر اساس میزان دقت کنش‌ها تعیین می‌شد[۲۲]. در دهه‌های اخیر این مدل از LCS در حل مسائل واقعی زیادی نقش داشته است[۲۳-۲۵]. هدف اصلی XCS ساخت یک نقشه کامل و دقیق از فضای مسئله است. در الگوریتم XCS، یک مسئله یادگیری به صورت یک مسئله انتخاب عمل تعریف می‌شود که عامل باید برای حل آن عملی را انتخاب کند. عامل با استفاده از مجموعه قوانین تصمیم‌گیری تلاش می‌کند بهترین عمل را برای مسئله پیدا کند. مجموعه [P] شامل قوانین دسته‌بندها یا مجموعه‌ای از عبارات شرط-عمل است. در هر دور از الگوریتم یک مجموعه قوانین تطبیق یافته [M] ساخته می‌شود و عامل با استفاده از یک الگوریتم یادگیری تقویتی سعی می‌کند بهترین عمل را از یک مجموعه کنش [A] برای مسئله انتخاب کند. در ادامه قوانین مربوط به عمل به‌روزرسانی می‌شوند. هدف از به‌روز رسانی قوانین، افزایش دقت و دانش عامل در مسئله است.

الگوریتم ۱، مراحل اجرای TinyXCS را نشان می‌دهد. در گام اول اطلاعات از محیط دریافت می‌شوند و با توجه به مجموعه P، مجموعه M را ایجاد می‌کند (خطوط ۱ و ۲). آرایه پیش‌بینی با توجه به رویکرد تعیین شده ایجاد می‌شود و در صورت نیاز عمل پوشش انجام می‌شود (خط ۳). در مرحله بعد، بهترین عمل دارای بیشترین ارزش انتخاب و به محیط اعمال می‌شود (خطوط ۴ و ۵). در نهایت با دریافت پاداش از محیط مقدار P محاسبه شده و مجموعه A-1 به‌روز می‌شود (خطوط ۶ تا ۸). برای تشخیص نیاز به استفاده از الگوریتم ژنتیک است(خطوط ۱۰ تا ۱۳). در انتها مقادیر A ذخیره شده و سپس برای گام‌های بعدی پاک می‌شود (خطوط ۱۳ و ۱۴). الگوریتم TinyXCS با بهره‌گیری از یادگیری تقویتی، ایجاد تعامل با محیط، دریافت پاداش و به‌کارگیری الگوریتم‌های تکاملی همچون ژنتیک، سعی در پیدا کردن دسته‌بند بهینه دارد. اما، باید توجه داشت که الگوریتم‌های فرااابتکاری به دلیل عملکرد تصادفی ممکن است از نقاط بهینه منحرف شوند. الگوریتم TinyDT با حذف شاخه‌های کم اهمیت و ایجاد محدودیت در عمق درخت، با وجود کاهش حافظه مصرفی موجب حذف برخی از حالات و تصمیمات سیستم می‌شود و در نتیجه، با افت دقت همراه است.

۳-۳- درخت تصمیم

درخت تصمیم یکی از روش‌های تحلیل و پردازش داده‌ها در حوزه یادگیری ماشین است. این مدل براساس ساختار شاخه‌بندی تصمیم‌ها و شرایط موجود، به صورت سلسله مراتبی عمل می‌کند[۲۶]. برای ساخت یک درخت تصمیم، ابتدا با بررسی مجموعه داده، بهترین معیار تقسیم را انتخاب و بر اساس این معیار، درخت را به شاخه‌های مختلف تقسیم می‌کنیم. این فرایند ادامه می‌یابد تا در نهایت به گره‌های برگ (نتایج نهایی) برسیم. درخت تصمیم نهایی قابلیت پیش‌بینی یا دسته‌بندی داده‌ها را بر اساس معیارهایی که در هنگام ساخت درخت تصمیم مشخص شده‌است دارد. چند نمونه پرکاربرد درخت تصمیم می‌توان به درخت تصمیم دودویی، درخت تصمیم چند مقطعی، درخت تصمیم یکپارچه، درخت تصمیم ادغام شده و درخت تصمیم تصادفی اشاره کرد[۲۷-۲۹].

[11] Leraning Classifier System

الگوریتم (۱) : الگوریتم TinyXCS برای توزیع بارکاری در لبه شبکه

Input: condition
$//\alpha(t), C(t), B(t)$
Output: action for workload allocation in Edge layer
$//\beta(t)$

```
input ← state of environment
M ← CreateMachSet(P)
PA ← BuildPredictionArray(M)
A ← BuildActionSet(action_selection_strategy, PA)
action ← SelectAction(A)
environment ← Perform(action)
environment ← Reward()
P = previous_reward + discount_factor
                    * max_prediction
UpdateSet(P, previous_action_set)
If (flag and UseGenericAlgorithm(action_set)) do
    GenericAlgorithm(A, previous_input)
End If
previous_action_set ← A
previous_reward ← environment of reward
```

الگوریتم (۲) : الگوریتم TinyDT برای توزیع بارکاری در لبه شبکه

Input: *data*
$//\alpha(t), \beta(t), C(t), B(t)$
Output: prediction and decision for workload allocation in Edge layer

```
TinyDT ← MaxDepth and MinSamples
//memory management parameters in tree
Class Node
//init value
    label ← label of Node
    children ← list of subtrees
    decision ← decision index
End Class
If (TinyDT) do
    For (x in data) do
        label = CreateLabel(x)
    End For
    Node(label) ← label
End If
best_decision_index ← FindingBestDesicion(data)
root ← Node(None)
root.decision ← best_decision_index
subtrees ← SplitDataSet(data, best_decision_index)
For (subtree in subtrees ) do
    child = DecisionTreeTrain(subtree, TinyDT)
    children.append(child)
    root ← children.value
End For
```

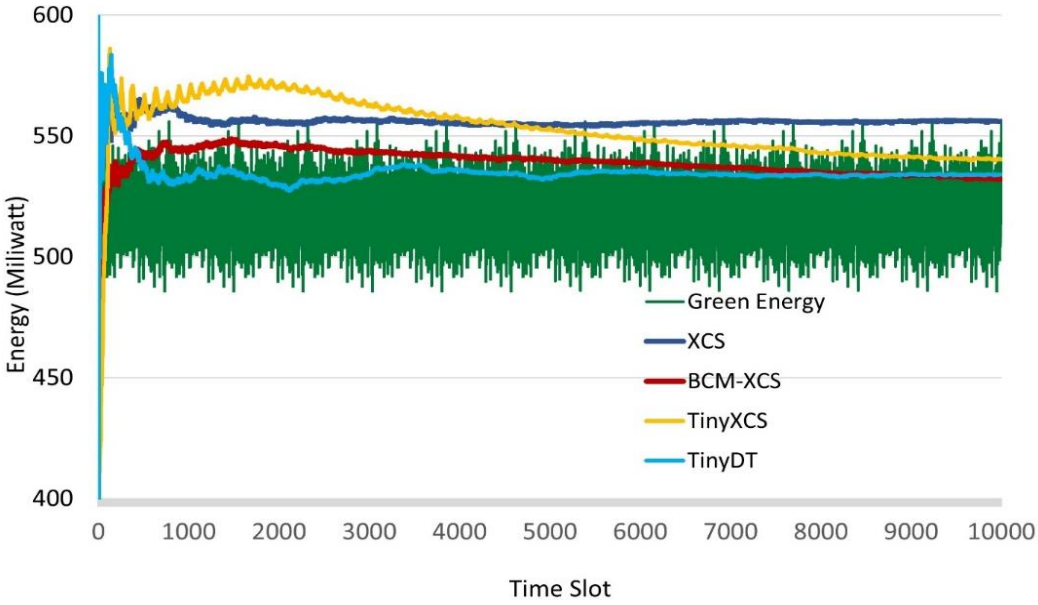
در الگوریتم ۲، درخت تصمیم برای توزیع بارکاری در لبه شبکه نشان داده شده است. در بخش اول الگوریتم، پارامترهای ورودی را برای آموزش مدل به درخت تصمیم می‌دهیم. این اطلاعات شامل بارکاری ورودی، ازدحام شبکه، میزان باتری و بارکاری پردازش شده در لبه شبکه برای پیش‌بینی و تصمیم‌گیری آینده سیستم است. درخت تصمیم پیشنهادی در این پژوهش باید از نظر مصرف حافظه بهینه و قابل مدیریت باشد. برای مدیریت مصرف حافظه در مدل پیشنهادی از پارامترهای MaxDepth و MinSamples برای مدیریت عمق درخت و حذف شاخه‌های کم اهمیت استفاده می‌کنیم(خط ۱). در ادامه مقادیر اولیه برای ساخت گره‌های درخت مورد نیاز را تعیین می‌کنیم(خطوط ۲ تا ۵). این مقادیر شامل برچسب گره، لیست فرزندان گره و شاخص تصمیم‌گیری است. با توجه به شرایط و محدودیت‌های در نظر گرفته شده برای درخت تصمیم و داده‌های موجود در مسئله، در هر مرحله برچسب گره‌های درخت با استفاده از تابع CreateLabel ساخته می‌شود(خطوط ۷ تا ۱۲). تابع FindingBestDecisin با توجه به اطلاعات دریافتی و شاخص‌های مورد اهمیت در مسئله سعی در پیدا کردن بهترین تصمیم در مدل پیشنهادی دارد(خط ۱۳). شاخص‌های اصلی ما در این پژوهش تأخیر و انرژی مصرفی کمینه است که تابع FindingBestDecisin رسیدگی به این موضوع را بر عهده دارد. در ادامه گره ریشه بدون برچسب و بعد از آن شاخص تصمیم‌گیری که توسط تابع ذکر شده در بالا تعیین شد مقداردهی می‌شود(خطوط ۱۴ و ۱۵). در نهایت با توجه به شاخص و داده‌های مسئله زیردرخت‌های دیگر با استفاده از تابع SplitDataSet در مراحل بعد ساخته می‌شوند(خط ۱۶). با توجه به زیر درخت‌های تولید شده و شاخص بدست آمده و همچنین محدودیت‌های تعیین شده برای ساخت درخت تصمیم کوچک(TinyDT)، فرزندان و ریشه‌های زیر درخت‌ها برای تمام داده‌های مسئله ساخته می‌شود (خطوط ۱۷ تا ۲۱). الگوریتم TinyDT به دلیل آگاهی از شرایط و حالت‌های مختلف سیستم می‌تواند به سرعت تصمیم‌گیری لازم برای توزیع بارکاری در لبه شبکه انجام دهد. با توجه به تفاوتهای فوق در عملکرد دو مدل، در کاربردهای واقعی با توجه به نیازها و اهداف مسئله یکی از آن‌ها را برای توزیع بار در لبه شبکه، در دستگاه سوییچ ایستگاه پایه می‌توان پیاده‌سازی کرد و نیازی به ترکیب این مدل‌ها نیست.

۴- پیاده‌سازی و ارزیابی

روش پیشنهادی TinyXCS روی سیستمی با پردازنده ۲ هسته‌ای و فرکانس ۱.۸ گیگاهرتز و حافظه ۲ گیگابایت با زبان C++ پیاده‌سازی شده‌است. همچنین روش پیشنهادی دوم یعنی TinyDT بر روی همین سیستم و با زبان پایتون پیاده‌سازی شده و روی مجموعه داده حاصل شده از روش BCM-XSC آموزش داده شده‌است. در این بخش ابتدا مقادیر اولیه اختصاص داده شده برای پارامترهای سیستم بیان شده و سپس نتایج حاصل ارزیابی مورد بررسی قرار می‌گیرد. در این سناریو، میزان بارکاری ورودی در هر بازه بین ۱۰ تا ۱۳۰ درخواست بر ثانیه است که به صورت تصادفی وارد شبکه می‌شود. ازدحام شبکه در بازه‌های زمانی مختلف بین ۱۰ تا ۲۰ میلی‌ثانیه متغیر است. ورودی انرژی تجدیدپذیر به توزیع نرمال با میانگین ۵۲۰ میلی‌وات و انحراف معیار ۱۵۰ میلی‌وات به سیستم وارد می‌شود. حداکثر ظرفیت باتری‌ها ۲۰۰۰ میلی‌وات تنظیم شده‌است که در لحظه ابتدایی ظرفیت باتری برابر صفر است. مصرف انرژی ثابت ایستگاه پایه ۳۰۰ میلی‌وات و حداکثر تعداد سرویس‌دهنده‌های لبه برابر ۱۰ است. هنگامی که سرویس‌دهنده‌ها فعال هستند، انرژی مصرفی هر یک از این سرویس‌دهنده‌ها ۱۵۰ وات است.

حداکثر نرخ پردازش هر سرویس‌دهنده ۲۰ درخواست بر ثانیه و هزینه عملیاتی هر واحد باتری ۰.۰۱ در نظر گرفته می‌شود. همچنین ضریب هزینه هر واحد از منابع تامین برق پشتیبان برابر ۰.۱۵ است. این شبیه‌سازی با استفاده از XCS پایه، BCM-XCS، TinyXCS و TinyDT انجام شده‌است.

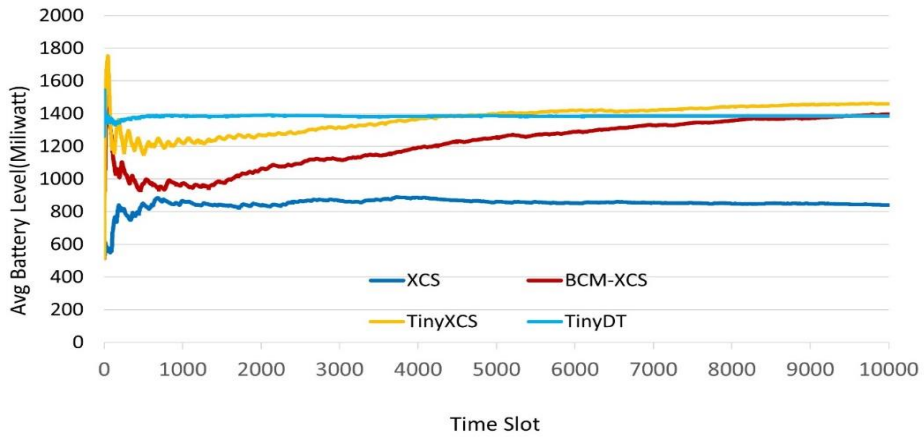
شکل ۴، میانگین مصرف انرژی سیستم در پردازش بارهای کاری و انرژی تجدیدپذیر را برای سه مدل LCS (XCS، BCM-XCS و TinyXCS) و مدل درخت تصمیم(TinyDT) بر حسب میلی‌وات در هر بازه زمانی نشان می‌دهد. همانطور که از شکل مشخص است، برای XCS پایه، میانگین توان مورد نیاز برای پردازش بار کاری‌ها در لبه، بیشتر از توان فراهم شده از منابع تجدیدپذیر است. بنابراین، می‌توان نتیجه گرفت که ترکیب انرژی باتری‌ها و منابع تجدیدپذیر برای پردازش بار کاری‌ها لازم است. از سوی دیگر، مصرف انرژی سیستم کنترل شده TinyXCS از بازه زمانی ۴۰۰۰ کمتر از سیستم کنترل شده XCS است و پس از گذشت مدت زمان بیشتر و یادگیری سیستم بهبود پیدا کرده و با اختلاف ۱۰ درصدی نسبت به BCM-XCS تقریباً برابر با میزان



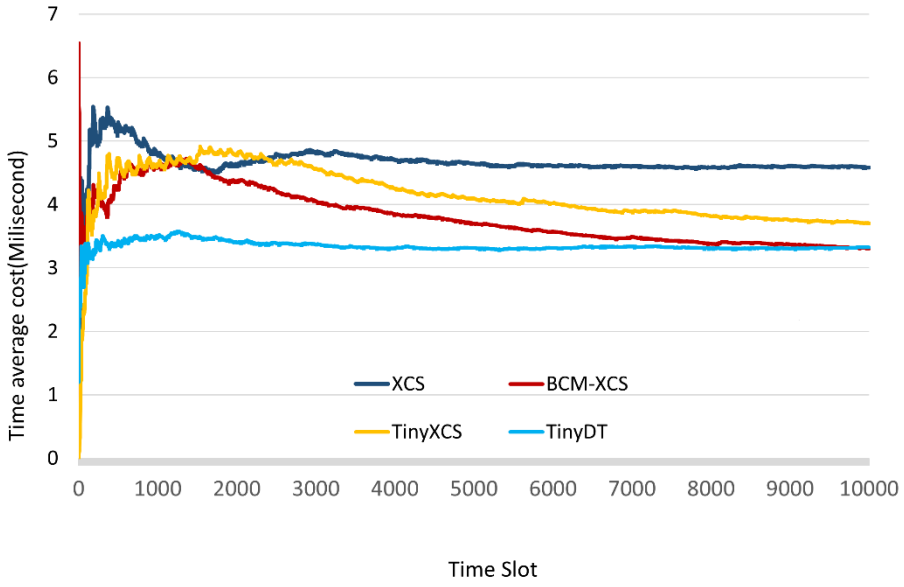
شکل ۴- میانگین مصرف انرژی برای سه مدل XCS، BCM-XCS و TinyXCS

انرژی تجدیدپذیر در لبه شبکه است و این مصالحه با توجه به کاهش ۵۰ درصدی حافظه در TinyXCS صورت گرفته‌است. بنابراین، می‌توان نتیجه گرفت که TinyXCS علاوه بر مصرف منابع کم در توزیع بهینه بارکاری موفق است، به‌طوری که انرژی مصرفی مورد نیاز بیشتر از منابع سبز تأمین می‌شود تا مصرف باتری به حداقل برسد. علاوه بر این همانطور که مشاهده می‌شود مدل TinyDT نسبت به همه مدل‌های XCS مصرف انرژی کمتری دارد و علت این امر آموزش برون‌خط سیستم و گرفتن بهترین تصمیمات در توزیع بارکاری است. در ابتدا و تا بازه زمانی ۵۰۰، انرژی مصرفی درخت تصمیمTinyDT بیشترین حد خود را داردکه علت آن ساخت درخت و انرژی مورد نیاز برای آن است. در ادامه، با گذشت زمان نمودار انرژی به صورت ثابت و با نوسان کمی حرکت می‌کند و درنهایت بیشترین حد انرژی با روش BCM-XCS برابر می‌شود. نکته قابل اهمیت در این نتایج، تأمین بیشترین انرژی مصرفی مورد نیاز سیستم از منابع سبز در این مدل است که موجب ذخیره حداکثری انرژی در باتری‌ها می‌شود. کنترل مصرف انرژی کاهش مصرف انرژی تجدیدپذیر در لبه شبکه را به دنبال دارد. که نتیجه آن ذخیره انرژی تجدیدپذیر بیشتر در باتری‌های لبه شبکه می‌شود.

شکل ۵ سطح باتری‌های چهار مدل XCS پایه، BCM-XCS، TinyXCS و TinyDT را نشان می‌دهد. نتایج حاصل از شبیه‌سازی نشان می‌دهد که سطح باتری در هر سه مدل XCS پس از مدت کمی شارژ می‌شود. نکته قابل توجه در مدل TinyXCS افزایش سطح باتری و شیب صعودی آن پس از ۱۰۰۰ بازه زمانی است که در انتهای اجرا به بیش از ۱۴۰۰ میلی‌وات می‌رسد. درصورتی‌که در مدل XCS پایه باتری پس از ۴۵۰۰ بازه زمانی به ۸۰۰ میلی‌وات همگرا شده و ثابت مانده است و در مدل BCM-XCS درنهایت به ۱۴۰۰ میلی‌وات می‌رسد. دلیل نوسانات اولیه خالی بودن حافظه‌های [P] است. در مدل XCS پایه به دلیل اضافه شدن دسته‌بندهای تصادفی در طول زمان، سطح باتری همگرا می‌شود. در مدل BCM-XCS با اضافه شدن دسته‌بندهای بهینه به حافظه BCM، احتمال انتخاب دسته‌بند بهینه از این حافظه بیشتر می‌شود. در صورتی که درمدل TinyXCS به دلیل محدود بودن ظرفیت حافظه مصرفی میزان ذخیره انرژی در باتری‌ها بیشتر است. در مدل TinyDT با توجه به تأمین انرژی مورد نیاز از منابع سبز باتری‌ها پس از گذشت بازه زمانی ۵۰۰ شارژ شده و با گذشت زمان به طور تقریبی در ۱۴۰۰ میلی‌وات ثابت می‌ماند و این به معنی مصرف حداقلی باتری در مدل درخت تصمیم است. مدل پیشنهادی TinyDT از دو مدل XCS پایه و حافظه‌دار عملکرد بهتری در مصرف و ذخیره باتری از خود نشان می‌دهد و همانطور که در شکل ۵ مشاهده می‌شود نسبت به مدل پیشنهادی TinyXCS از بازه زمانی ۵۰۰۰ به بعد در ذخیره و شارژ باتری موفق‌تر عمل کرده که این نشان دهنده یادگیری پیوسته سیستم با گذشت زمان است.



شکل ۵- میانگین سطح باتری در لبه شبکه برای سه مدل XCS، BCM-XCS و



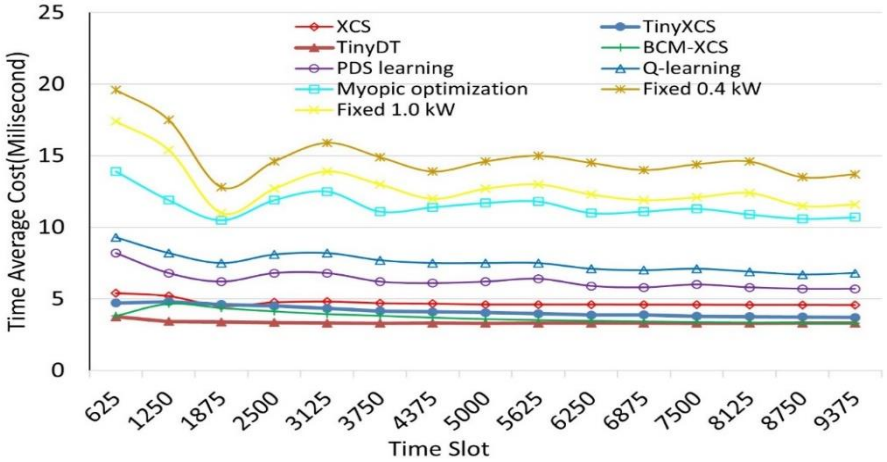
شکل ۶- هزینه تأخیر بارهای کاری در لبه شبکه برای سه مدل XCS، BCM-XCS و TinyXCS

شکل۶ میانگین تأخیر حاصل از شبیه‌سازی سیستم طبقه‌بند یادگیر را در ۱۰۰۰۰ بازه زمانی برای چهار مدل XCS، BCM-XCS و TinyXCS و TinyDT برحسب میلی‌ثانیه نشان می‌دهد. به دلیل پر نبودن حافظه‌ها در سه مدل XCS، ۱۰۰۰ بازه زمانی اول دارای نوساناتی هستند که با گذشت زمان و پر شدن حافظه‌ها، از شدت این نوسانات کاسته می‌شود. با افزایش تعداد بازه زمانی، سیستم‌های مبتنی بر LCS با توجه به پاداشی که از محیط دریافت می‌کنند گزینه بهتری جهت اعمال به محیط را انتخاب می‌کنند. پس از مدتی درXCS، حافظه P از دسته‌بندهای تصادفی پرشده و درنتیجه فرآیند یادگیری کندتر و تأخیر بارها پس از بازه زمانی ۴۰۰۰ ثابت می‌شود. بنابراین پس از بازه زمانی ۴۰۰۰ میانگین تأخیرها در XCS پایه، با شیب ملایمی کاهش‌یافته و به ۴/۵ میلی‌ثانیه همگرا می‌شود. درصورتی‌که در دو مدل BCM-XCS و TinyXCS پس از ۱۵۰۰ بازه زمانی با ورودی‌های جدید و با ادامه یادگیری میانگین تأخیرها با شیب نزولی کاهش‌یافته و در انتهای شبیه‌سازی برای هر مدل به ترتیب به ۳/۲ و ۳/۷ میلی‌ثانیه می‌رسد که با توجه به مصالحه انجام شده بین مصرف حافظه و تأخیر و کاهش ۵۰ درصدی حافظه در مدل پیشنهادی، اختلاف تأخیر ۰/۰۰۵ درصدی بین دو مدل قابل قبول است. در مدل TinyDT با توجه به اینکه سیستم بهترین تصمیم را با توجه به ویژگی‌های مستخرج از مجموعه داده می‌گیرد، شاهد تأخیر کمتری نسبت به دیگر مدل‌ها هستیم. همانطور که در شکل ۶ مشاهده می‌شود تأخیر مدل درخت تصمیم با گذشت زمان کاهش

از محدودیت‌های روش برخط استفاده از الگوریتم‌های فراابتکاری است که به دلیل خاصیت تصادفی، می‌توانند از نقاط بهینه خارج شوند. همچنین در یافته و در نهایت برابر با تأخیر مدل BCM-XCS می‌شود و به مقدار۴/۵ میلی‌ثانیه میل می‌کند.

شکل ۷ میانگین هزینه تأخیر پردازش بارها در لبه شبکه و ارسال باقی آن‌ها به لایه بالایی را برای ۱۰ هزار بازه زمانی برای الگوریتم‌های مختلف توزیع بارکاری نشان می‌دهد. با توجه به شکل، روش‌های توان ثابت و Myopic به دلیل عدم پیش‌بینی تأخیرها در ابتدای شبیه‌سازی تأخیر زیادی را متحمل می‌شوند، اما روش‌های PDS و Q-Learning [۳۰, ۳۱]و روش‌های مبتنی بر LSC به دلیل یادگیری و پیش‌بینی شرایط، تأخیر به نسبت کمتری دارند که در میان این روش‌ها، دو مدل BCM-XCS و TinyXCS دارای تأخیر بسیار کمتری هستند. همانطور که در شکل ۶ مشاهده می‌شود از بین روش‌های بررسی شده، سه روش BCM-XCS، TinyXCS و TinyDT حداقل هزینه تأخیر را دارند. مدل TinyXCS به عنوان یک مدل برخط برای توزیع بارکاری در لبه شبکه با وجود کاهش ۵۰ درصدی حافظه مصرفی نتایج مشابهی با مدل حافظه‌دار BCM-XCS دارد که به دلیل پاداش دریافتی از محیط پس از مدتی شاهد روند کاهشی هزینه تأخیر در این مدل هستیم. همچنین مدل TinyDT به عنوان یک مدل برون‌خط و آگاه از آینده سیستم، با توجه به شرایط تعریف شده برای مسئله بهترین تصمیم را اتخاذ می‌کند که نتیجه آن کمترین هزینه تأخیر نسبت به باقی روش‌های توزیع بار است.

روش برون‌خط با حذف شاخه‌های کم‌اهمیت، ممکن است دقت سیستم تصمیم‌گیرنده کاهش یابد.



شکل ۷- مقایسه میانگین هزینه تأخیر در روش‌های مختلف توزیع بارکاری

۵- نتیجه‌گیری

انتقال حجم زیاد داده‌ها به ابر-مه باعث تأخیر طولانی در پردازش بارهای کاری می‌شود. این مقاله روشی را برای بهینه سازی توزیع بارهای کاری و ایجاد تعادل بین تأخیر و مصرف انرژی و منابع محاسباتی در لبه پیشنهاد می‌کند. در روش پیشنهادی ابتدا از یک LCS به نام TinyXCS و سپس از یک مدل درخت تصمیم کوچک شده به نام TinyDT استفاده شد. نتایج نشان می‌دهد که تأخیر در پردازش بارهای کاری و سطوح باتری در مدل TinyXCSپیشنهادی نسبت XCS بهبود داشته و تقریبا با BCM-XCS قابل رقابت است. میانگین هزینه تأخیر TinyXCS پس از ۱۰۰۰ بازه زمانی شروع به کاهش می‌کند و به ۳ میلی‌ثانیه نزدیک می‌شود که تقریبا معادل روش BXC-XCS و بهتر از هر روش‌های دیگر توزیع بارکاری است. همچنین روش پیشنهادی TinyDT با حافظه مصرفی حداکثر ۳ کیلوبایت و هزینه تأخیر حداکثر ۴ میلی‌ثانیه بهترین نتیجه را در بین سایر مدل‌های مشابه دارد.

با توجه به پیدایش پردازش‌های درون-شبکه و اهمیت آنها در تسریع محاسبات توزیع شده در لبه شبکه، پیاده‌سازی مدل‌های TinxXCS و TinyDTدر پردازنده‌های شبکه‌ی تعبیه شده در سوییچ‌های لبه شبکه جهت درپردازش بلادرنگ درخواست‌های توزیع بار کاری را به عنوان ادامه این کار می‌توان در نظر گرفت.

مراجع

[1] G. Li, J. Yan, L. Chen, J. Wu, Q. Lin, and Y. Zhang, "Energy consumption optimization with a delay threshold in cloud-fog cooperation computing," *IEEE access*, vol. 7, pp. 159688-159697, 2019.

[2] X. Niu, S. Shao, C. Xin, J. Zhou, S. Guo, X. Chen, *et al.*, "Workload allocation mechanism for minimum service delay in edge computing-based power internet of things," *IEEE Access*, vol. 7, pp. 83771-83784, 2019.

[3] M. Abbasi, M. Yaghoobikia, M. Rafiee, A. Jolfaei, and M. R. Khosravi, "Efficient resource management and workload allocation in fog–cloud computing paradigm in IoT using learning classifier systems," *Computer Communications*, vol. 153, pp. 217-228, 2020.

[4] M. Abbasi, E. M. Pasand, and M. R. Khosravi, "Workload allocation in iot-fog-cloud architecture using a multi-objective genetic algorithm," *Journal of Grid Computing*, pp. 1-14, 2020.

[5] M. Abbasi, M. Yaghoobikia, M. Rafiee, M. R. Khosravi, and V. G. Menon, "Optimal distribution of workloads in cloud-fog architecture in intelligent vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, pp. 4706-4715, 2021.

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, pp. 2347-2376, 2015.

[7] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, pp. 1-20, 2016.

[8] S. H. Shah and I. Yaqoob, "A survey: Internet of Things (IOT) technologies, applications and challenges," *2016 IEEE Smart Energy Grid Engineering (SEGE)*, pp. 381-385, 2016.

[9] A. Ometov, O. L. Molua, M. Komarov, and J. Nurmi, "A survey of security in cloud, edge, and fog computing," *Sensors*, vol. 22, p. 927, 2022.

[10] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, pp. 1587-1611, 2013.

[11] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing," in *2017 Global Internet of Things Summit (GIoTS)*, 2017, pp. 1-6.

[12] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the internet of things," in *2017 IEEE international conference on edge computing (EDGE)*, 2017, pp. 17-24.

[13] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, pp. 637-646, 2016.

[14] C.-H. Hong and B. Varghese, "Resource Management in Fog/Edge Computing: A Survey," *arXiv preprint arXiv:1810.00305*, 2018.

[15] Dennis, D. K. a. Gaurkar, Y. a. Gopinath, S. a. Goyal, S. a. Gupta, C. a. Jain, *et al.* (2022, 2017/9/2). *EdgeML: Machine Learning for resource-constrained edge devices (0.4 ed.)*. Available: https://github.com/Microsoft/EdgeML

[16] س. قاسمی فلاورجانی, م. نعمت بخش and ب. شاهقلی قهفرخی, "تخصیص وظایف چندهدفه در واگذاری به ابر سیار," مجله مهندسی برق دانشگاه تبریز, vol. 46, pp. 217-232, 2016.

[17] و. ستاری نائینی, ی. سالم and ع. راشدی, "بهره‌گیری از الگوریتم پرش ترکیبی قورباغه جهت کاهش مصرف انرژی مراکز داده ابری از طریق بهینه‌سازی مدیریت زمان‌بندی کارها و ترکیب مؤثر ماشین‌های مجازی," مجله مهندسی برق دانشگاه تبریز, vol. 48, pp. 687-698, 2018.

[18] H. Wu, L. Chen, C. Shen, W. Wen, and J. Xu, "Online geographical load balancing for energy-harvesting mobile edge computing," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1-6.

[19] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, and C. Liu, "Fog computing for energy-aware load balancing and scheduling in smart factory," *IEEE Transactions on Industrial Informatics*, vol. 14, pp. 4548-4556, 2018.

[20] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "Generalization in the XCSF Classifier System: Analysis, Improvement, and Extension," *Evol. Comput.*, vol. 15, pp. 133-168, 2007.

[21] J. Holland, L. Booker, M. Colombetti, M. Dorigo, D. Goldberg, S. Forrest, *et al.*, "What Is a Learning Classifier System?," in *Learning Classifier Systems*. vol. 1813, P. Lanzi, W. Stolzmann, and S. Wilson, Eds., ed: Springer Berlin Heidelberg, 2000, pp. 3-32.

[22] S. W. Wilson, "Classifier fitness based on accuracy," *Evol. Comput.*, vol. 3, pp. 149-175, 1995.

[23] B. Bartin, "Use of learning classifier systems in microscopic toll plaza simulation models," *IET Intelligent Transport Systems*, vol. 13, pp. 860-869, 2019.

[24] M. R. Karlsen and S. Moschoyiannis, "Evolution of control with learning classifier systems," *Applied network science*, vol. 3, p. 30, 2018.

[25] M. H. Arif, J. Li, M. Iqbal, and K. Liu, "Sentiment analysis and spam detection in short informal text using learning classifier systems," *Soft Computing*, vol. 22, pp. 7281-7291, 2018.

[26] E. Alpaydin, *Introduction to Machine Learning*, 3 ed. Cambridge, MA: MIT Press, 2014.

[27] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, pp. 255-260, 2015.

[28] B. De Ville and P. Neville, *Decision trees for analytics: using SAS Enterprise miner*: SAS Institute Cary, NC, 2013.

[29] P.-N. T. M. S. Vipin, "Introduction to data mining," ed, 2006.

[30] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, pp. 361-373, 2017.

[31] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *Robotica*, vol. 17, pp. 229-235, 1999.