Computer Science Review 60 (2026) 100887

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Revisiting workflow scheduling with the power of edge computing: Taxonomy, review, and open challenges

Shenghai Li[a,b], Wentai Wu[c], Haotong Zhang[d], Yongheng Liu[b], Weiwei Lin[e,b,*], Keqin Li[f]

[a] School of Future Technology, South China University of Technology, Guangzhou, 510000, China
[b] Pengcheng Laboratory, Shenzhen, 518000, China
[c] Department of Computer Science, College of Information Science and Technology, Jinan University, Guangzhou, 510632, China
[d] School of Software Engineering, South China University of Technology, Guangzhou, 510006, China
[e] School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China
[f] State University of New York, New Paltz, New Paltz, New York, 12561, USA

A R T I C L E   I N F O

Keywords:
Edge computing
Workflow scheduling
Task dependency
Taxonomy

A B S T R A C T

Edge computing has emerged as a pivotal paradigm for overcoming the limitations of traditional cloud computing, especially in latency-sensitive applications such as autonomous driving and video streaming. As mobile applications grow in complexity, they often consist of interdependent tasks that can be modeled as workflows. Scheduling these workflows over heterogeneous resources at the network edge presents unique challenges due to the diverse characteristics of workflows and the complex nature of edge environments. Despite recent advances, a comprehensive overview of the fundamentals and state-of-the-art approaches in this field remains lacking. This survey systematically reviews workflow scheduling in edge computing by first addressing its motivation, typical application scenarios, and core challenges. The survey then introduces basic models and performance metrics, followed by a taxonomy of existing scheduling strategies categorized by research issues, optimization objectives, and techniques. Finally, we discuss open challenges and propose future research directions, providing a guide for the development of efficient edge workflow scheduling strategies.

1. Introduction

The proliferation of demanding smart applications, such as AR/VR, autonomous driving, and industrial IoT [1–3], strains the capabilities of resource-constrained mobile devices [4,5] and challenges traditional cloud computing models due to high latency and network congestion [6,7]. Edge computing offers a compelling solution by distributing resources closer to end-users [8], thereby enabling low-latency communication and efficient local processing which are crucial for meeting the stringent Quality of Service (QoS) requirements of these applications.

However, beyond latency, the increasing complexity and multi-stage nature of many smart applications present significant scheduling challenges, especially in the unique context of the edge. For example, applications in areas such as real-time video analytics pipelines, multi-step industrial automation, complex event processing, and distributed AI inference often consist of interdependent tasks with inherent sequential or parallel execution constraints. Scheduling these complex applications monolithically often overwhelms the limited capacity of individual edge servers, creating resource bottlenecks and leading to inefficient utilization within the heterogeneous and resource-constrained edge environment [9,10]. Conversely, decomposing them into simple independent tasks while ignoring their critical interdependencies risks violating execution constraints and can lead to incorrect results or outright application failure.

To effectively manage such structured applications, modeling them as workflows, typically represented as Directed Acyclic Graphs (DAGs)—becomes essential. This workflow abstraction allows for fine-grained, dependency-aware scheduling, enabling the optimization of task placement, data movement, and resource allocation across distributed edge nodes while respecting execution order constraints. Consequently, developing specialized workflow scheduling strategies tailored to the unique challenges and characteristics of edge computing, distinct from simpler task placement or traditional cloud-based approaches, is a critical research imperative necessary for unlocking the performance potential of complex edge applications.

* Corresponding author at: School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China.
Email addresses: lishh01@pcl.ac.cn (S. Li), wentaiwu@jnu.edu.cn (W. Wu), sezhanght@mail.scut.edu.cn (H. Zhang), liuyh01@pcl.ac.cn (Y. Liu), linww@scut.edu.cn (W. Lin), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.cosrev.2025.100887
Received 10 June 2025; Received in revised form 20 November 2025; Accepted 27 December 2025

1.1. Core challenges

Edge computing offers a promising solution to address the growing demand for low-latency, energy-efficient, and scalable computing. However, the unique system characteristics of edge computing, e.g., resource constraints, heterogeneity, and dynamic network conditions, pose significant challenges for the overall performance of edge systems and greatly impact on workflow scheduling. In this section, we will delve into these characteristics.

1.1.1. Resource constraints

Edge servers typically have limited computational power, storage, and bandwidth [9], which directly affect workflow scheduling. Restricted resources prolong task execution and data transmission, impairing overall performance [10]. Limited capacity also constrains parallel task execution [11], making it difficult to handle complex workflows efficiently. Consequently, workflow scheduling must incorporate resource management policies that adapt to changing demands and prevent performance bottlenecks.

1.1.2. Heterogeneity

Edge environments encompass diverse nodes with varying computational, storage, and energy capabilities. Workflows themselves are heterogeneous, containing tasks with different resource requirements and QoS demands. This dual heterogeneity intensifies the complexity of workflow scheduling: a node well-suited for one task may be ill-equipped for another. For instance, a high-computation node may lack sufficient storage for data-intensive tasks, while a storage-rich node may not meet strict latency requirements. Such multi-dimensional heterogeneity necessitates scheduling algorithms that judiciously match tasks to nodes, ensuring that both the edge infrastructure's diversity and the workflow's varying needs are addressed.

1.1.3. Distributed nature

Edge computing's distributed architecture reduces latency by placing resources closer to data sources. However, this dispersion complicates workflow scheduling. Tasks within a workflow often depend on each other's outputs and may need to run on different nodes. Ensuring correct task sequencing across distributed nodes is non-trivial, and cross-node data transfers introduce additional latency and overhead. Workflow scheduling in such contexts must manage these spatial dependencies and communication costs, synchronizing task execution to maintain end-to-end efficiency.

1.1.4. Dynamic network

Edge computing systems often rely on wireless networks, which can be subject to interference, channel stochasticity, and signal attenuation [12], leading to fluctuating bandwidth, intermittent connectivity, and unpredictable delays. Since workflow tasks frequently exchange data, unstable networks can disrupt the entire execution chain. Scheduling decisions must anticipate network variability, balancing task placement and transmission paths to preserve the workflow's integrity and timeliness. Algorithms need to be robust, reacting swiftly to degraded links and adjusting resource allocations or task migration strategies as needed.

1.1.5. Mobility

User mobility introduces additional complexity to workflow scheduling. As users move between coverage areas of different edge servers, tasks in the workflow may need to be relocated to maintain connectivity [13]. This relocation is not a simple reassignment of isolated tasks since workflows encompass multiple interdependent tasks whose execution order and data dependencies must be preserved. When a user's movement causes the connected edge node for one task to change, it can trigger a cascade of adjustments throughout the workflow.

1.1.6. Scalability

Since edge infrastructures can scale to thousands of heterogeneous nodes, maintaining a coherent global view across such a dispersed network imposes heavy communication and coordination overhead, frequently yielding outdated or incomplete state information. Moreover, allocating large DAG-structured workflows to an ever-growing pool of resources is challenging, with the scheduling decision space growing rapidly as task and node counts increase. Together, these factors erode scheduler responsiveness and QoS guarantees at scale, making scalability an essential challenge for workflow scheduling in production-grade edge environments. Collectively, these characteristics distinguish edge computing from traditional cloud environments, rendering cloud-centric scheduling approaches often inadequate and underscoring the need for specialized workflow scheduling strategies designed for the unique constraints and dynamics of the edge.

1.2. Related surveys

Review of edge computing: Hong and Varghese [14] provided a comprehensive review of resource management within fog/edge computing, focusing on the unique challenges of managing distributed resources characterized by resource constraints, heterogeneity, and system dynamics. Covering studies from 2013 to 2018, the article categorizes architectures, infrastructure, and underlying algorithms. Similarly, Luo et al. [15] conducted an extensive survey on resource scheduling in edge computing, emphasizing optimized performance in the evolving IoT and wireless networks landscape. This survey explores detailed edge computing architectures, discusses key research issues—computation offloading, resource allocation, and resource provisioning—and classifies various scheduling techniques based on operation modes. Additionally, it summarizes crucial performance metrics and highlights the importance of resource scheduling through diverse application scenarios.

Sahni et al. [16] presented a survey on Distributed Resource Scheduling (DRS) within edge computing, addressing motivations, challenges, and existing solutions. This survey distinguishes DRS challenges from those in traditional parallel and distributed systems, emphasizing the motivations and potential enabled by DRS. It introduces a taxonomy of existing literature based on systems, problems, and solution approaches. Xia et al. [17] investigated resource management in emerging UAV-enabled Edge Computing (UEC), introducing a conceptual UEC architecture, discussing collaboration and communication models, and providing a taxonomy of existing resource management strategies.

Zhang and Debroy [18] offered an in-depth review of resource management within Mobile Edge Computing (MEC), focusing on the diverse performance requirements of user applications and the dynamic nature of MEC environments. The authors categorize existing solutions into conventional optimization-based approaches and emerging learning-based methods. Additionally, this study delves into prevalent application cases in MEC, particularly video analytics, identifying the pipelined nature of such applications and discussing DAG-based workflow scheduling. However, it lacks a systematic analysis and review of workflow scheduling.

Review of workflow scheduling: Wu et al. [19] conducted a thorough survey of workflow scheduling in cloud environments, emphasizing the challenges and significance of workflow scheduling in the cloud. The authors provide a taxonomy of workflow scheduling strategies based on resource information and workflow characteristics and comprehensively discuss ten critical workflow scheduling problems with respective cloud-based solutions. Similarly, Adhikari et al. [20] reviewed workflow scheduling in cloud computing, covering workflow model architecture, classification, and management systems. This survey examines various scheduling methods, categorizes them by objectives and techniques, and explores emerging trends in serverless and fog computing. Hosseinzadeh et al. [21] reviewed multi-objective optimization techniques for independent task/workflow scheduling in cloud environments, focusing on

Table 1
Summary of related surveys and our work.

Paper	Year	Task relations	Environment	Methodological focus	Perspective
[14]	2019	Independent	Fog/Edge	Resource management	Review of resource management, focusing on architecture, infrastructure, and underlying algorithms.
[15]	2021	Independent	Edge	Comprehensive scheduling	Survey of resource scheduling: architecture, key issues, techniques, metrics, applications.
[16]	2022	Independent	Edge	DRS-centric	Analysis of DRS, highlighting challenges, motivations, solutions, and future directions.
[17]	2022	Independent	Edge	UAV-specific strategies	Review of UAV-enabled Edge Computing: architectures, collaboration models, resource strategies.
[18]	2023	Independent	Edge	Optimization vs. Learning	In-depth MEC review: optimization-based versus learning-based approach for resource management.
[19]	2015	Workflow	Cloud	Problem based taxonomy	Survey of workflow scheduling in cloud: challenges and ten classic scheduling problems.
[20]	2019	Workflow	Cloud	Taxonomy by objectives and techniques	Workflow techniques in cloud: definitions, taxonomy, and emerging serverless/fog trends.
[21]	2020	Both	Cloud	Multi-objective meta-heuristics	Multi-objective optimization for independent tasks/workflows: meta-heuristics and limits.
[22]	2024	Workflow	Cloud-Fog	Comprehensive scheduling	A clear taxonomy based on techniques, metrics, dependencies, policies, and evaluation tools.
[23]	2024	Workflow	Cloud-Fog	AI-based and Heuristic Approach	Systematic review focusing specifically on AI (75%) and Heuristic (25%) techniques for workflow scheduling in cloud-fog environment. Identifies serverless/FaaS as a key research gap.
[24]	2024	Workflow	Cloud-Edge	RL-based Scheduling	Taxonomy and review specifically for RL-based scheduling. Proposes an RL-centric taxonomy (based on agent architecture, RL algorithms, etc.) and discusses RL-specific open challenges.
Ours	2025	Workflow	Cloud-Edge	Comprehensive scheduling	Comprehensive survey of edge-centric workflow scheduling: motivations, models, metrics, techniques, future directions.

Acronyms used in this table: Distributed Resource Scheduling(DRS), Reinforcement Learning(RL).

meta-heuristic multi-objective optimization schemes and analyzing their characteristics and limitations.

More recently, several highly relevant surveys have been presented. Bouabdallah and Fakhfakh [22] presented a Systematic Literature Review of workflow scheduling in Cloud-Fog environments, proposing a comprehensive taxonomy of scheduling techniques, metrics, dependencies, policies, and evaluation tools. Concurrently, other surveys have adopted a more methodology-specific focus. Khaledian et al. [23] provided a systematic review specifically for AI-based and heuristic algorithms, also within the Cloud-Fog context, finding that 75% of recent works use AI or hybrid methods. Similarly, Jayanetti et al. [24] offered a deep-dive review and taxonomy focused exclusively on reinforcement learning techniques for Cloud-Edge environments. Their work details an RL-centric classification covering agent architectures and algorithms, alongside RL-specific open challenges.

Comparison to our work: As summarized in Table 1, existing surveys have bifurcated their focus, addressing either independent task scheduling in edge environments or workflow scheduling in traditional cloud environments. While recent surveys [22–24] have begun to address workflow scheduling in hybrid systems, their analyses remain specialized. These works are limited either in scope, by focusing on the Cloud-Fog paradigm [22,23], or in methodology, by concentrating exclusively on a single technique such as AI-based and heuristic approaches [23] or reinforcement learning [24].

Consequently, a comprehensive survey covering the full spectrum of scheduling techniques for Edge-centric environments has been absent. This survey bridges this gap by providing a holistic review of workflow scheduling in Edge-centric systems. We synthesize the full range of methodological approaches—spanning mathematical programming, heuristic/meta-heuristic and RL-based methods. This work presents a multi-perspective taxonomy covering system models, research issues, performance metrics, scheduling pattern simulators, datasets, and optimization techniques, culminating in a discussion of open research challenges.

1.3. Contribution and organization

The main contributions of this article are as follows:

- The motivation for workflow scheduling in edge computing is discussed, and core challenges that highlight research needs are outlined.
- A presentation of typical scenarios demonstrating the application potential of this domain is provided, followed by the formulation of the basic model of edge workflow scheduling.
- A multidimensional taxonomy of existing workflow scheduling strategies in edge computing is provided, categorized by key research issues, optimization objectives, scheduling patterns, and optimization techniques, as well as the simulation environment and datasets used for evaluation.
- Open challenges and future directions of workflow scheduling within edge computing are explored, offering insights for further studies in the field.

The rest of the article is organized as follows: Section 2 details the systematic methodology used to conduct this review, including the literature search strategy and screening criteria. Section 3 provides background information, covering the basic model of workflow scheduling in edge computing and application scenarios. Sections 4 and 5 collectively present a comprehensive taxonomy of existing workflow scheduling strategies in edge computing, categorizing them based on key research issues, optimization objectives, optimization approaches, along with the simulation environment and datasets used for evaluation. Section 6 discusses open challenges and future directions in this field. The conclusion of this article is presented in Section 7.

2. Methodology

2.1. Literature search strategy and data sources

This review is based on a systematic and reproducible literature search. The Web of Science (WoS) Core Collection was selected as the primary data source, chosen for its comprehensive index of high-impact, peer-reviewed journals and conference proceedings in computer science.

The search strategy targeted the Topic (TS) field—which includes the title, abstract, and keywords—to ensure comprehensive retrieval. The query combined key synonyms for the target platforms (e.g., edge, fog,

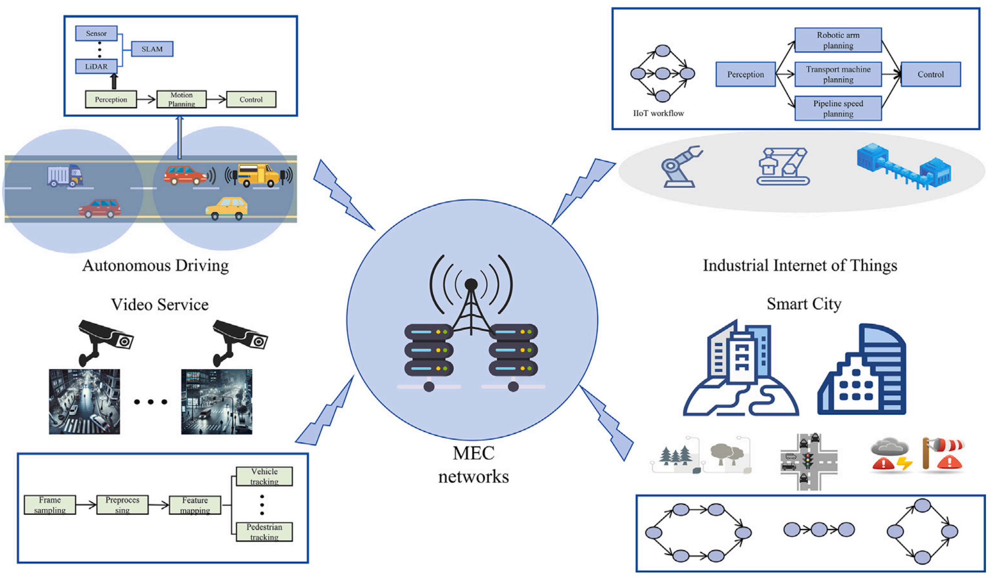

Fig. 1. Various application scenarios for workflow scheduling in edge computing.

cloud-edge) and the core problem domain (e.g., workflow, dependent tasks). The exact query string was:

```
TS=((''edge computing'' OR ''fog computing'' OR ''cloud-
edge collaboration'' OR ''cloud-fog collaboration'')
AND (''workflow scheduling'' OR ''dependent task
scheduling''))
```

The search was limited to publications from January 1, 2019, to the present, yielding an initial corpus of 422 publications. The annual distribution of these articles, depicted in Fig. 2, illustrates the field's research momentum. A significant upward trend is revealed, with annual publications growing steadily from 27 in 2019 to 117 in 2024. This growth indicates that the domain has recently become a major research focal point. The 50 publications recorded for 2025, representing a partial count for the current year, further confirm sustained research interest.

In addition to the temporal trend, Fig. 3 illustrates the distribution of these publications by their primary research focus. The analysis reveals a clear concentration on Edge (283 papers), constituting the vast majority of the literature at 62.3%. This is followed by Fog/Cloud-Fog (113 papers), accounting for 26.8% of the corpus, while Cloud-Edge (26 papers) represents a more specific focus on the collaboration between the cloud and edge tiers, comprising 10.9%. This distribution indicates that edge-centric solutions are the dominant focus of the research community. These numerical analyses, showing both a rapid escalation in volume and a strong focus on edge-centric challenges, underscore the pressing need for a systematic review to structure and synthesize the current state-of-the-art. These 422 publications served as the initial document pool for screening against the inclusion and exclusion criteria detailed in Section 2.2, which derived the final set of papers for in-depth analysis.

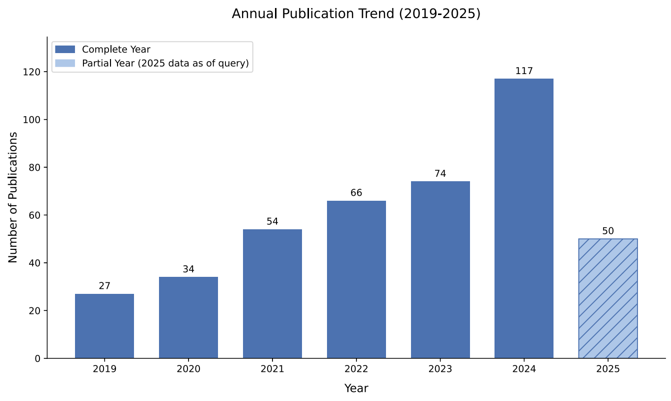

Fig. 2. The annual publication trend of the retrieved literature (2019–2025). Data sourced from Web of Science Core Collection. **Note:* The 2025 data represents a partial count for the year.

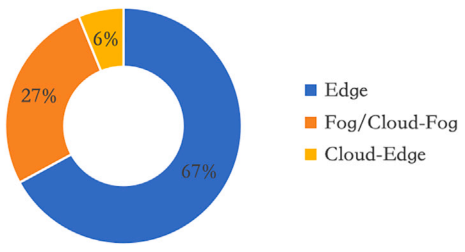

Fig. 3. Proportion of literature related to the focus environment.

2.2. *Screening and inclusion criteria*

The initial search yielded 422 potentially relevant publications. To identify the core literature for this review, we employed a multi-stage screening process designed to assess relevance and methodological rigor. This process was guided by the following inclusion and exclusion criteria:

— **Inclusion criteria:** (i) The paper must propose, evaluate, or analyze a specific workflow scheduling strategy, algorithm, or framework explicitly for edge/cloud-edge environments; (ii) the paper must address task dependencies (e.g., workflows or DAGs), while studies focusing solely on independent task offloading were excluded; (iii) published between January 1, 2019, and the present, with 2025 data being partial; (iv) published as a peer-reviewed journal article or conference paper; and (v) written in English.

— **Exclusion criteria:** (i) Papers focusing solely on cloud/fog computing without an edge component; (ii) non-archival publications, such as short papers (typically less than 6 pages), posters, abstracts, editorials, and tutorials; (iii) papers where the full text was not accessible; (iv) duplicate publications of the same study; and (v) publications that are themselves literature reviews, surveys, or meta-analyses, as this review focuses on primary studies.

The screening was conducted in multiple stages, beginning with a review of titles and abstracts, followed by a full-text assessment of the remaining candidates. Furthermore, this database search was supplemented by a snowballing process [25], where the reference lists of all included papers were manually scanned to identify additional relevant studies that might have been missed by the initial query.

This combined methodology resulted in a final corpus of 153 core publications that forms the basis of the taxonomy and review presented in Sections 4 and 5.

3. Background

This section provides background on edge workflow scheduling. It begins with illustrative application scenarios, then details the fundamental models for workflows and edge systems.

3.1. Application scenarios

In this section, we discuss several representative application scenarios including IIoT, video services, autonomous driving, and smart cities, as shown in Fig. 1. These application scenarios exemplify the specific benefits of workflow scheduling in edge computing, highlighting its role in enabling efficient, reliable, and scalable solutions across diverse industries.

3.1.1. Autonomous driving

Autonomous driving, a transformative transportation technology, is widely regarded as a promising solution to alleviate traffic congestion and enhance travel safety [26]. Autonomous driving systems involve several key tasks, including environment perception, planning, decision-making, and execution [27]. These tasks are interdependent, with planning and decision-making relying on the output of environment perception, followed by the execution of control instructions. Exploring deeper, parallel execution within these modules reveals more intricate dependencies. For example, the environment perception module can process data streams from sensors, LiDAR, and GPS in parallel. Upon completion of these parallel data processing tasks, the results are integrated to perform simultaneous localization and mapping (SLAM).

Moreover, the tasks involved in autonomous driving are both data- and computation-intensive. For example, the perception module processes massive amounts of data, and precise decision-making relies heavily on complex deep learning (DL) models [28,29]. While fundamental for accuracy, these DL models, e.g., for real-time object detection and path planning, are notoriously computation-intensive. Furthermore, training these models requires diverse data from vast fleets of vehicles, which raises significant data privacy and communication bottleneck issues. To address these training challenges, federated learning (FL) is being pervasively applied. FL enables collaborative model training on vehicles without centralizing sensitive raw data, thus preserving privacy [30,31]. This introduces a dual challenge: the demand for high-performance computation for real-time DL inference and the need for coordinated resource management for distributed FL training. Given these resource demands, edge computing emerges as an ideal solution, offering real-time data processing and powerful computational capabilities. Consequently, workflow scheduling in edge computing can greatly enhance the performance of autonomous driving systems by ensuring efficient task execution and optimal resource allocation, which has been explored in recent studies [32,33].

3.1.2. Industrial Internet of Things

The Industrial Internet of Things (IIoT) refers to the integration of IoT technologies in industrial manufacturing systems. In a typical IoT scenario, numerous smart devices and sensors are interconnected, generating vast amounts of data. These data streams are often complex and require rapid processing to ensure the performance, reliability, and safety of IIoT systems [34]. For example, in a smart factory, real-time data from manufacturing equipment and assembly lines are continuously generated, with predictive analytics used to forecast equipment failure or optimize production scheduling. Such real-time data streams necessitate edge computing for low-latency processing. In IIoT systems, dependencies between tasks are prevalent. Typical examples involve anomaly detection and pipeline speed control, where anomaly detection algorithms depend on the sensor data from equipment in real time, while pipeline speed control algorithms adjust production rates based on the system's current status. Efficient workflow scheduling in edge computing environments is essential to ensure these tasks are executed in the correct sequence and within time constraints, thereby optimizing system performance and enhancing operational efficiency. Several related researches have made contributions to advancing this application scenario [35–37].

3.1.3. Video service

As a typical computation-intensive and data-intensive service, video service requires significant computation and storage resources [38], and introduces great network pressure since it typically involves streaming large amounts of video data [39]. Edge computing presents an ideal paradigm in this context by enabling data processing closer to the source, which significantly enhances the efficiency of tasks such as transcoding and content caching [40]. This proximity to data sources further alleviates network congestion, ensuring a seamless, high-quality user experience. Furthermore, workflow scheduling is crucial in managing task dependencies within video services, where critical tasks such as video capturing, encoding, and content delivery are often interdependent. For example, Xie et al. [41] described a workflow in cloud–edge environments for video surveillance, where the complete service is achieved through a series of steps. These steps include capturing video data through cameras, detecting motion, identifying objects, tracking these objects, and adjusting camera angles via pan-tilt-zoom (PTZ) controls as needed. Rong et al. [42] discussed a camera stream workflow applied in large-scale video analytics within an edge-cloud environment. A complete camera stream workflow consists of capturing, frame resizing and sampling, feature mapping, and object tracking and detection. These tasks exhibit a clear dependency structure, where each task relies on the output of the preceding task. Additionally, all tasks except capturing can be distributed across different compute nodes to meet specific latency and accuracy requirements, optimizing resource allocation and ensuring efficient task execution throughout the workflow.

3.1.4. Smart city

Smart city aims to enhance the quality and efficiency of urban services by integrating information and communication technologies [43]. Typical urban services, such as traffic management, environmental monitoring, and public safety, often involve processing vast amounts of data collected from a wide range of sources, including sensors, cameras, and intelligent devices. Driven by this data-intensive nature, edge computing becomes essential for real-time data processing, reducing latency, and ensuring the responsiveness of various services. In the context of smart cities, various modern applications comprise complex and independent tasks. For example, in intelligent traffic flow management, real-time data from cameras and sensors must be processed to analyze traffic conditions and predict congestion. This data then informs adaptive traffic signal control systems that dynamically adjust traffic flow based on the current situation. Environmental monitoring tasks often involve collecting data from various sensors (e.g., air quality, temperature, humidity), which can be processed in parallel to detect patterns or anomalies. This analysis can trigger actions such as adjusting energy consumption or initiating emergency responses. Thus, a smart city presents a typical and complex application scenario for workflow scheduling in edge computing.

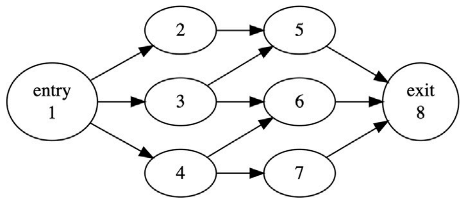

Fig. 4. An example of a workflow modeled as a DAG.

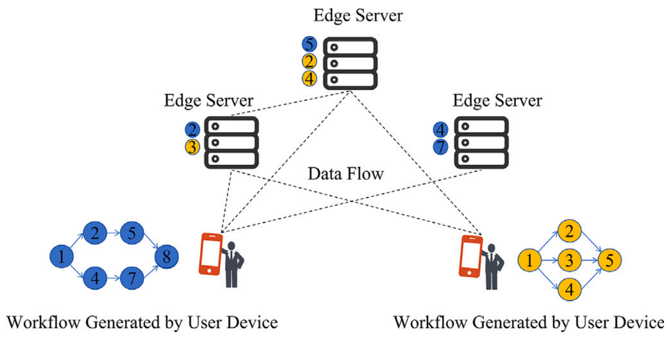
Edge Server
Edge Server
Edge Server
Data Flow
Workflow Generated by User Device
Workflow Generated by User Device

Fig. 5. An example of task offloading.

3.2. Basic model

A workflow consists of a sequence of interdependent tasks executed in a specific order, which is commonly modeled as a DAG, $G = \{V, E\}$. Within this model, each vertex in V indicates a task, and each edge in E represents a dependency between tasks. For example, in Fig. 4, the workflow consists of eight tasks, $V = \{T_1, T_2, \dots, T_8\}$, with dependencies such as $E = \{E_{T_1,T_2}, E_{T_1,T_3}, \dots, E_{T_5,T_8}\}$. An edge $E_{T_i,T_j} \in E$ signifies that T_j is dependent on T_i, where T_j cannot be executed before T_i has been completed. With $E_{T_i,T_j} \in E$, task T_i is defined as a predecessor of task T_j and task T_j is defined as a successor of task T_i. The relationships of tasks in workflows can be classified into two categories according to their dependencies: sequential and parallel. Sequential tasks, such as $< T_2, T_5, T_8 >$ in Fig. 4, exhibit direct or indirect dependencies and must be executed in a precise sequence. In contrast, parallel tasks, e.g., $< T_2, T_3, T_4 >$ in Fig. 4, have no such dependencies and may be executed in any order. Additionally, the entry and exit dummy tasks are typically placed at the start and end of the workflow, marking initiation and termination, respectively.

Each task $T_i \in V$ in a workflow can be characterized by several attributes, i.e., $T_i = \{D_i, C_i, R_i, Pred_{T_i}, Succ_{T_i}\}$, where D_i stands for the data size of T_i, C_i is the computational load of T_i(usually measured in CPU cycles), R_i represents the data size of the output produced by T_i, which also corresponds to the weight of the edge connecting T_i to its successor task(s), $Pred_{T_i}$ denotes the set of predecessor tasks of T_i and $Succ_{T_i}$ represents the set of successor tasks of T_i.

An edge computing system is generally a two-tier architecture comprised of an edge layer and a user device layer. The user layer is made up of diverse user devices, ranging from the ubiquitous mobile phones and tablets to more specialized equipment such as intelligent sensors deployed in smart factories, devices with high mobility like Unmanned Aerial Vehicles (UAVs) and vehicles, and other Internet of Things (IoT) devices. Each device generates requests, performs basic processing, and submits complex processing requests to the edge layer. The edge layer comprises a set of heterogeneous servers that are geographically distributed and vary in their capabilities and resource availability. Each edge server E_k can be denoted as $E_k = \{CPU_k, MEM_k, BW_k\}$, where CPU_k refers to the computational power (in terms of CPU cycles per second), MEM_k denotes the available memory and BW_k indicates the available network bandwidth. Connected via advanced wireless technologies, the edge layer supports the simultaneous interconnection of numerous user devices and efficient data transmission, enabling real-time processing.

Based on the task and system models defined above, the core workflow scheduling problem is to find an optimal execution plan—determining *where* to execute each task T_i (i.e., on which server E_k) and *when* to start its execution (ST_i).

4. Taxonomy

In this survey, we provide a taxonomy of existing workflow scheduling strategies in edge computing, categorized by research issues, optimization objectives, scheduling patterns, and optimization techniques, as well as the simulation environments and datasets used for evaluation, as shown in Fig. 6.

4.1. Research issues

In a typical workflow scheduling scenario at the edge, task offloading, resource allocation, and service caching/placement are three major research issues.

4.1.1. Task offloading

Task offloading, as shown in Fig. 5, refers to transferring the task from user devices to edge servers for remote processing. An offloading strategy determines which tasks to execute locally and which to offload to edge servers.

The offloading decision hinges on various factors, e.g., task workload, the capacity of the user device, network conditions, and the current load on the edge servers.

Assume that N edge servers are deployed and workflows are generated by user devices. Tasks in a workflow will then be processed locally or offloaded to edge servers. Let $Off(T_i)$ denote the offloading decision, which is expressed as

$$Off(T_i) = \begin{cases} 0, & \text{task process locally} \\ m, & \text{task offloaded to edge server m, } m \in \{1, 2, .., N\}. \end{cases} \quad \text{s.t.} \quad \text{C1, C2} \tag{1}$$

where:

$$\text{C1 (Uniqueness Constraint):} \quad \sum_{m=0}^{N} \mathbb{I}(Off(T_i) = m) = 1$$

$$\text{C2 (Assignment Precedence):} \quad \forall T_j \in \mathrm{Pred}(T_i), \quad Off(T_j) \in \{0, 1, \dots, N\}$$

Task dependencies within a workflow restrict the order in which tasks must be executed, introducing a layer of complexity to task offloading. This dependency constraint can lead to idle time in the execution queue when a task ready for execution must wait for its dependent tasks to complete. Thus, it is vital to employ an effective strategy to determine the best execution order, ensuring that the execution queue meets the dependency restrictions while minimizing the idle time of the execution queue. Intuitively, the offloading decision-making process may involve strategies such as offloading serial tasks to the same server to maintain execution flow and distributing parallel tasks across multiple servers to leverage concurrent processing capabilities. Moreover, task dependencies affect the data flow within the workflow, as the output of one task often serves as the input for successor tasks. Consequently, offloading decisions must also factor in data transfers; reducing data transfer times is crucial for optimizing overall workflow execution time. For example, it is advisable to prioritize offloading tasks with high data volume to servers with high bandwidth.

As a fundamental and essential component of workflow scheduling, the vast majority of related studies consider task offloading as their primary research issue.

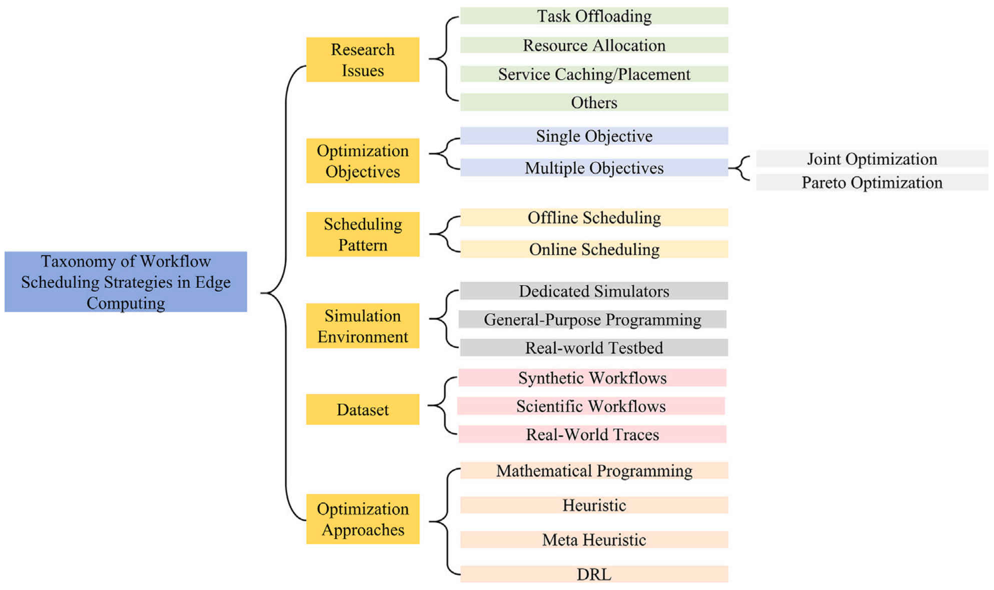

Fig. 6. Taxonomy of workflow scheduling strategies in edge computing.

4.1.2. Resource allocation

Following the offloading decision, resource allocation is a pivotal step in workflow scheduling for edge computing. This process involves allocating the necessary resources such as CPU and bandwidth, to the offloaded tasks on edge servers. Numerous studies have been dedicated to the optimization of resource allocation to enhance workflow performance and efficiency [35,44–47].

Resource allocation aims to optimize the resource utilization at the edge while ensuring that each task's requirements are adequately met. The resource allocation strategy must consider the current availability of resources on each edge server and the QoS requirements of offloaded tasks. For instance, a task with a tight deadline should be allocated with abundant computational resources. For a task T_i that has been offloaded to server k, the resource allocation decision A_{T_i} can be expressed as $A_{T_i} = \{CPU_{T_i}, MEM_{T_i}, BW_{T_i}\}$, where CPU_{T_i}, MEM_{T_i}, and BW_{T_i} represent the number of CPU cycles, memory, and network bandwidth allocated to T_i, respectively.

The allocation decision for server k is primarily governed by resource capacity constraints, which dictate that the total resources consumed by concurrently executing tasks must not exceed the server's maximum capacity. Formally, at any given time t:

$$\sum_{T_i \in \text{Active}(k,t)} CPU_{T_i} \le CPU_k \quad \wedge \quad \sum_{T_i \in \text{Active}(k,t)} MEM_{T_i} \le MEM_k \\ \wedge \quad \sum_{T_i \in \text{Active}(k,t)} BW_{T_i} \le BW_k$$

Here, $\text{Active}(k,t)$ denotes the set of tasks actively executing and consuming resources on server k at time t. The allocation must also adapt dynamically to the changes in task demands and server states. For example, if a server becomes overloaded or a task's requirements change due to evolving conditions at the user layer, the allocation strategy must be able to respond accordingly. Furthermore, in scenarios where multiple tasks compete for the same resources simultaneously, the allocation strategy must prioritize tasks based on criteria such as urgency, importance, or the predicted time required for task completion.

The allocation must also adapt dynamically to the changes in task demands and server states. For example, if a server becomes overloaded or a task's requirements change due to evolving conditions at the user layer, the allocation strategy must be able to respond accordingly. Furthermore, in scenarios where multiple tasks compete for the same resources simultaneously, the allocation strategy must prioritize tasks based on criteria such as urgency, importance, or the predicted time required for task completion.

In the context of workflow scheduling, managing workflow dependencies is critical to ensure that resources are optimally allocated. By analyzing the dependency graph of tasks in a workflow, the strategy identifies tasks on the critical path and ensures that these tasks receive priority in resource allocation. This approach reduces delays in workflow completion by minimizing the risk of critical tasks that directly affect the makespan of the entire workflow being hindered by resource scarcity. By prioritizing tasks based on their importance and position in the workflow, the resource allocation strategy improves available resource utilization and facilitates the efficient execution of workflows.

4.1.3. Service caching/placement

In edge computing, service caching/placement plays a pivotal role in the efficient scheduling of workflows, as they directly affect task execution latency and overall system performance, which has been extensively investigated in several studies [48–50]. Workflow scheduling often involves a series of interdependent tasks that may require frequent access to shared data or services. By strategically placing services and caching data at edge nodes, the data retrieval time between workflow tasks can be minimized, thereby improving the execution time of the entire workflow. However, due to the constrained and heterogeneous nature of resources at the edge, determining optimal service placement remains a complex challenge.

Consider a workflow W composed of T tasks, where each task $T_j \in W$ may need access to a specific service S_i that can be cached at an edge node $k \in E$. The binary decision variable $x_{i,k}$ indicates whether service i is cached at node k:

$$x_{i,k} = \begin{cases} 1, & \text{if service } i \text{ is cached at node } k, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

An optimal service caching/placement decision can effectively reduce response latency and improve user experience. However, service caching/placement is subject to strict resource constraints:

$$\sum_{i \in S} x_{i,k} \cdot r_i \le C_k, \quad \forall k \in E, \tag{3}$$

where r_i denotes the resource requirement for caching service i, and C_k is the total resource capacity of node k.

Beyond task offloading, resource allocation, and service caching/placement, there are several other research issues, such as fault tolerance in unreliable environments and network routing path selection for data transmission.

4.2. Optimization objectives

Optimization objectives in existing workflow scheduling strategies can be broadly categorized into three types: single-objective optimization, joint optimization, and Pareto optimization. Single-objective optimization focuses on a single performance metric, while joint optimization and Pareto optimization address multiple metrics simultaneously.

4.2.1. Single-objective optimization

Single-objective optimization strategies aim to enhance a specific performance metric, such as reducing latency, minimizing energy consumption, or lowering costs, while adhering to specific system constraints.

Makespan. Makespan is the most widely adopted metric, as it directly impacts user QoE by determining how quickly results are delivered. A vast body of research focuses on its minimization [11,51–65]. It is formally defined as the time elapsed from the start of the entry task T_{entry} to the completion of the exit task T_{exit}:

$$Makespan(W) = FT(T_{exit}) - ST(T_{entry}). \tag{4}$$

The finish time $FT(T_i)$ of any task T_i is the sum of its start time $ST(T_i)$ and execution time $ET(T_i)$. The start time itself is constrained by the task's arrival time, resource availability, and the data dependencies on its predecessor tasks $T_j \in Pred(T_i)$, including transmission times ($Time_{trans}^{ji}$):

$$ST(T_i) = \max\left(Time_{arrival}^{i}, \max_{T_j \in Pred(T_i)} \left(FT(T_j) + Time_{trans}^{ji}\right), Time_{available}^{m}\right). \tag{5}$$

Researchers have applied this model in various contexts. For instance, Liu et al. [53] optimized the average makespan across multiple workflows, while Kanemitsu et al. [51] focused on minimizing makespan under strict energy constraints for mobile clients.

Energy. Energy optimization is critical for prolonging device longevity and enhancing system-wide energy efficiency [66–68]. A common approach defines the total execution energy consumption $E(W)$ consumed by a workflow W as the sum of the energy consumed by its individual tasks T_i:

$$E(W) = \sum_{T_i \in W} E(T_i). \tag{6}$$

Each task's energy consumption $E(T_i)$ can be broken down into components associated with local device processing and edge server processing:

$$E(T_i) = E_{\text{local}}(T_i) + E_{\text{edge}}(T_i). \tag{7}$$

The local energy consumption depends on whether the task T_i is executed locally ($Off(T_i) = 0$) or offloaded to an edge server S_m ($Off(T_i) = m > 0$):

$$E_{\text{local}}(T_i) = \begin{cases} E_{\text{execution}}^{\text{local}}(T_i) + \sum_{T_j \in \text{Pred}(T_i)} E_{\text{download}}(T_j, T_i) & \\ \quad + \sum_{T_k \in \text{Succ}(T_i)} E_{\text{upload}}(T_i, T_k), & \text{if } Off(T_i) = 0, \\ E_{\text{upload}}^{m}(T_i), & \text{if } Off(T_i) = m > 0\,. \end{cases} \tag{8}$$

Here, $E_{\text{execution}}^{\text{local}}(T_i)$ is the energy for local execution. $E_{\text{download}}(T_j, T_i)$ and $E_{\text{upload}}(T_i, T_k)$ represent the energy consumed for receiving data from predecessor T_j and sending data to successor T_k, respectively. If T_i is offloaded, $E_{\text{upload}}^{m}(T_i)$ is the energy used to transmit its input data to edge server S_m.

Similarly, if the task is offloaded ($Off(T_i) = m > 0$), the energy consumed by the edge server S_m includes receiving the task's input data, executing the task, and managing data transfers with predecessors and successors located elsewhere:

$$E_{\text{edge}}(T_i) = E_{\text{download}}(T_i) + E_{\text{execution}}^{m}(T_i) + \sum_{T_j \in \text{Pred}(T_i)} E_{\text{download}}(T_j, T_i) + \sum_{T_k \in \text{Succ}(T_i)} E_{\text{upload}}(T_i, T_k). \tag{9}$$

While Eqs. (8)–(9) effectively model the dynamic energy consumption, a comprehensive system-level assessment, crucial for edge and fog environments, must also account for the energy consumed during inactive periods. Specifically, idle energy consumption, the power drawn by devices including edge servers and user terminals while powered on but not actively processing or transmitting data for this workflow, can constitute a significant portion of the total energy budget [69].

Therefore, a more holistic system-level energy model, $E_{\text{system}}(W)$, integrates both dynamic and idle components. A simplified representation can be expressed as:

$$E_{\text{system}}(W) = E(W) + \sum_{k \in D} P_{\text{idle},k} \times T_{\text{idle},k} \tag{10}$$

where $E(W)$ represents the total dynamic energy associated with task execution and data transmission, derived from Eqs. (8)–(9). The second term estimates the total idle energy consumed system-wide. Within this term, D denotes the set of all devices participating in the workflow's execution, $P_{\text{idle},k}$ is the average idle power consumption specific to device $k \in D$, and $T_{\text{idle},k}$ represents the estimated total idle duration for device k during the workflow's makespan $Makespan(W)$. This idle duration $T_{\text{idle},k}$ is calculated by subtracting the total time device k spends actively executing tasks and communicating data related to workflow W from the overall makespan M.

Several studies have focused on optimizing the energy consumption of the entire system. For instance, Chakraborty and Mazumdar [67] focused on energy-efficient task offloading in sensor-based MEC environments. Few works have concentrated on the energy usage of specific key components. Hu et al. [66], for example, explored adaptive scheduling in vehicular edge environments, specifically minimizing the energy usage by roadside units. To make the optimization problem more aligned with practical service requirements, more recent studies also incorporate makespan constraints, balancing energy savings with task deadlines [68].

Cost. Cost optimization, aimed at reducing user expenses while maintaining QoS, is another primary objective [46,70–74]. The total cost $Cost(W)$ is typically defined as the sum of computation, energy, and communication costs:

$$Cost(W) = Cost_{comp}(W) + Cost_{energy}(W) + Cost_{comm}(W), \tag{11}$$

where:

- $Cost_{comp}(W)$: Computation cost, calculated by multiplying execution time by resource unit costs.
- $Cost_{energy}(W)$: Energy cost, derived from total energy consumption multiplied by unit energy cost.
- $Cost_{comm}(W)$: Communication cost, based on data transmission volume multiplied by unit transmission cost.

Specific implementations of this model vary. Lin et al. [71] developed a scheme to minimize total execution cost under deadline constraints. Tang et al. [72] notably included financial penalties for SLA violations in their cost model. Zhang et al. [46] examined dynamic scheduling in collaborative edge-cloud environments, incorporating the cost of task migration.

Profit. From the service provider's perspective, profit optimization is the main goal [75]. Profit is typically modeled as the total price charged to the user minus the provider's operational cost:

$$Profit(W) = Price(W) - Cost(W). \tag{12}$$

Load balance. Closely related to utilization, Load Balance (LB) assesses the fairness and efficiency of workload distribution, which is crucial for preventing resource bottlenecks and improving responsiveness [76]. It is often measured as the standard deviation of load (U_i) across resources:

$$LB = \sqrt{\left(\sum_{i=1}^{N} (U_i - U_{\text{avg}})^2\right)/N}. \tag{13}$$

A lower LB value indicates a more balanced and desirable workload distribution.

Success rate. The Success Rate is a key Quality of Service (QoS) metric measuring end-to-end performance by the proportion of workflows that meet their deadlines [77–81]. This metric, S, is formulated as:

$$S = \left(\sum_{i=1}^{N} \delta_i\right)/N, \tag{14}$$

where N is the total number of workflows and δ_i is a binary indicator of whether workflow i completed on time.

Reliability. Reliability modeling in scheduling focuses on quantifying and mitigating the impact of component failures. This is critical in distributed systems where both computing nodes and network links are prone to faults [82,83]. A key challenge is to quantify this failure proneness and incorporate it into the optimization process.

A common strategy is to model reliability based on the historical failure rates (λ) of individual components, with a representative example of this approach provided by Asghari Alaie et al. [82]. In their work, reliability is modeled using an exponential decay function based on the component's failure rate and its active time. The reliability of executing a task t_i on a VM_p is defined as:

$$R(t_i, VM_p) = e^{-\lambda_{VM(p)} \cdot ET(t_i, VM_p)} \tag{15}$$

Similarly, the reliability of a communication link used for data transfer $e(t_j, t_i)$ is defined as:

$$R(e(t_j, t_i)) = e^{-\lambda_{L(p,q)} \cdot TT(t_j, t_i)} \tag{16}$$

The total reliability for a single task $Rel(t_i)$ is the product of its execution reliability and the reliability of all its prerequisite data links.

This task-level reliability model is then aggregated to represent the entire workflow. A common formulation defines the total workflow reliability, $Rel(W)$, as the product of the reliability of all its constituent tasks [84]:

$$Rel(W) = \prod_{t_i \in W} Rel(t_i). \tag{17}$$

Following this model, several studies have addressed workflow scheduling in unreliable environments with a focus on execution reliability, formulating problems that maximize reliability under constraints including deadlines and energy consumption [85,86].

Security. Security is a critical non-functional requirement focusing on the Confidentiality, Integrity, and Availability (CIA) triad [87]. In scheduling models, security is typically addressed using two primary strategies: as a hard scheduling constraint or as a quantifiable risk to be optimized. The constraint-based approach defines binary rules, such as constraining confidential tasks to private nodes [88,89]. A more flexible strategy models security as a quantifiable financial risk. A representative example is the Advanced Mean Failure Cost (AMFC) model [90], which aggregates threat probabilities to calculate an expected annual financial loss.

4.2.2. *Joint optimization*

Joint optimization generally considers and balances the effect of multiple performance metrics on the overall performance by employing a weighted sum of them. The objective function in joint optimization can typically be formulated as:

$$Min/Max\ F(x) = \alpha \times f_1(x) + \beta \times f_2(x) + \cdots + \gamma \times f_n(x), \tag{18}$$

where $f_n(x)$ represents a distinct performance metric, and α, β, …, and γ are the weight coefficients for different performance metrics, respectively. This weighted sum method effectively reflects and balances the importance of multiple performance indicators. Moreover, these weights can be adaptively adjusted to adapt to the dynamic environment, ensuring that the optimization process remains aligned with real-time demands. There have been many researches focusing on the joint optimization for workflow scheduling in edge computing [36,41,76,91–101].

Several studies have concentrated on optimizing makespan and energy consumption. For instance, Yan et al. [96] examined dependent task scheduling in MEC, jointly reducing makespan and energy consumption of user devices. Similarly, Fang et al. [98] explored this problem with a time-varying wireless fading channel. System-wide energy optimization is another focus. Xiao et al. [97] aimed to minimize both latency and energy consumption across the edge system. In urban vehicular networks, Zhao et al. [101] worked on optimizing average task delay and energy use for vehicles and RSUs.

Cost considerations have also been integrated into joint optimization. Xie et al. [41] focused on optimizing makespan and cost in cloud-edge environments, while Xie et al. [36] targeted minimizing makespan, energy consumption, and cost within IIoT-based edge computing.

Lu et al. [76] discussed the importance of load balancing to effectively utilize resources and improve the overall efficiency of the MEC system. Subsequently, they focused on jointly optimizing the energy consumption, cost, and load balance in the task scheduling process. Considering the transmission failure probability during dependent task scheduling in MEC, Al-Habob et al. [94] investigated the optimization problem of jointly minimizing makespan and failure probability.

4.2.3. *Pareto optimization*

Pareto optimization represents another critical field in multi-objective optimization, typically formulated with an objective function presented as follows:

$$Min/Max\ F(x) = \{f_1(x), f_2(x), \ldots, f_n(x)\} \tag{19}$$

Unlike single-objective or joint optimization, Pareto optimization seeks a set of solutions rather than a single numerical optimum. A solution is considered Pareto optimal if it is not dominated by any other solution, i.e. being superior on at least one objective and at least equal to it on other objectives. The goal of Pareto optimization is to find the set of all Pareto optimal solutions, which is called the Pareto front [102].

This method is particularly suitable for problems with conflicting performance metrics, such as balancing makespan and cost in workflow scheduling. Edge servers with more powerful resources typically reduce makespan but incur higher costs. Pareto optimization provides a set of solutions that simultaneously optimize key performance metrics [35,47,66,103–109].

Several studies have applied Pareto optimization to edge workflow scheduling. Hu et al. [66] and Cui et al. [105] addressed dependent task scheduling in MEC, optimizing makespan and user device energy consumption. Xu et al. [107] explored multi-objective workflow scheduling in SDN-based edge systems to optimize both makespan and system energy consumption. Peng et al. [103] and Song et al. [108] formulated a workflow offloading problem in MEC, simultaneously optimizing the makespan, cost, and energy consumption of the user device. For collaborative cloud-edge environments, Li et al. [104] proposed containerized workflow scheduling to minimize the makespan, load imbalance, and energy consumption. Considering the tradeoff between QoS and system efficiency, Kuang et al. [47] focused on reducing makespan, deadline violations, and the number of applied virtual machines (VMs) simultaneously.

4.3. Scheduling pattern

Based on when scheduling decisions are made, workflow scheduling approaches are broadly categorized into two patterns: offline and online.

4.3.1. Offline scheduling

In the offline scheduling pattern, all information about the workflow, including the DAG structure, task execution times, data dependencies, and resource availability, is assumed to be known in advance [35]. The goal of the scheduler is to compute a complete and static mapping of all tasks to resources before the first task begins execution. This approach is well-suited for repetitive, predictable applications where the environment is stable. It allows for the use of complex, time-intensive optimization algorithms (e.g., metaheuristic) to find a globally optimal or near-optimal solution. However, its static nature makes it unsuitable for highly dynamic edge environments where task arrivals are unpredictable or resource availability fluctuates.

4.3.2. Online scheduling

In the online scheduling pattern, decisions are made at runtime as new workflows or tasks arrive, or as the state of the edge environment changes [76,110]. Typically, this scheduling pattern has incomplete information about the future and must make decisions quickly based on the current state of the system, which is essential for the dynamic and unpredictable nature of edge computing. This requirement for rapid, real-time decision-making places a high demand on the response speed and low computational complexity of the scheduling algorithm.

4.4. Simulation environment

The validation of scheduling algorithms is predominantly conducted through simulation, as building and managing large-scale, reproducible real-world edge testbeds is complex and costly.

4.4.1. Dedicated simulators

A significant number of studies leverage established simulation toolkits. CloudSim [111] is a foundational, event-driven simulator widely used for modeling IaaS cloud environments, resource provisioning, and VM allocation. To better support workflow-specific research, WorkflowSim [112] was developed as an extension of CloudSim. It introduces a workflow engine, a failure model, and support for DAG-based dependencies, making it highly suitable for scientific workflow experiments. For environments with a fog or edge hierarchy, IFogSim [113], also based on CloudSim, is frequently used. It allows researchers to model the multi-layer IoT-Edge-Cloud architecture, assess network latency between layers, and evaluate energy consumption.

4.4.2. General-purpose programming

For highly novel or customized scheduling models that do not fit existing simulators, many researchers build their own discrete-event simulators using general-purpose languages. Languages such as Python, Java, and MATLAB are commonly used due to their extensive libraries for data structures, mathematical operations, and plotting. This approach offers maximum flexibility but requires significant implementation effort.

4.4.3. Real-world testbeds

A smaller number of studies validate their algorithms on real-world testbeds to provide the most accurate performance evaluation [114,115]. These testbeds often consist of heterogeneous single-board computers (e.g., Raspberry Pi) to emulate edge nodes, combined with commercial cloud VMs (e.g., Amazon EC2) to represent the cloud layer. While providing high fidelity, these results are often difficult to reproduce at scale.

4.5. Dataset

The choice of workload is critical for evaluating the performance and scalability of a scheduling algorithm. The datasets used in the literature can generally be divided into three categories.

4.5.1. Synthetic workflows

Synthetic workflows are algorithmically generated DAGs. Researchers typically define parameters such as the total number of tasks, the dependency structure, and the Communication-to-Computation Ratio (CCR), etc. The main advantage of synthetic data is the ability to conduct controlled experiments, allowing researchers to test the algorithm's performance under a wide variety of conditions and scales that may not be available in real-world traces.

4.5.2. Scientific workflows

This category includes standardized DAGs derived from real-world scientific applications. These are widely used because they represent complex, realistic and reproducible task dependencies. Prominent examples, many of which are utilized within the Pegasus workflow management system [116], include Montage (astronomy), LIGO (physics), Epigenomics (biology), and CyberShake (seismology). These datasets provide a common baseline for comparing the performance of a workflow scheduler.

4.5.3. Real-world traces

To evaluate schedulers in realistic, dynamic online scenarios, researchers often use traces from large-scale production data centers. The most commonly used traces are the Google Cluster Trace [117] and the Alibaba Cluster Trace [118]. These traces provide real-world data on task arrivals, execution times, resource demands, and dependencies over long periods, offering the highest fidelity for evaluating how an online scheduler performs under a real-world workload.

4.6. Approach

The approaches for solving edge workflow scheduling problems can be classified into four primary categories based on their underlying optimization techniques.

- **Mathematical Programming:** Formulates the scheduling problem as a formal optimization model, such as Integer Linear Programming (ILP) or Mixed-Integer Nonlinear Programming (MINLP), to find provably optimal or near-optimal solutions, though often at a high computational cost.
- **Heuristic Approaches:** Employ low-complexity, rule-based strategies, such as list scheduling and greedy algorithms, to find good, feasible solutions quickly, making them suitable for dynamic environments.
- **Meta-heuristic Approaches:** Utilize high-level, iterative strategies inspired by natural phenomena, including Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), to effectively explore large and complex solution spaces.
- **Deep Reinforcement Learning (DRL) Approaches:** Leverage learning-based agents, such as Deep Q-Networks (DQN) and Actor-Critic methods, that train on environmental feedback to develop adaptive policies capable of making rapid, intelligent decisions in dynamic and unpredictable systems.

Table 2
Review of mathematical programming–based works in edge workflow scheduling.

Paper	Year	Issue	Opt. objective	Sched. pattern	Simulation environment	Dataset	Approach	Highlights & limitations
[44]	2020	◇♣	ms & c	Off	CloudSim	Syn	Gradient Descent Search	Effective offline scheduling for a single workflow; lacking support for concurrent workflows adaptation.
[45]	2020	◇♣	ms & ec	Off	–	Syn	ILP-based Optimization	Decoupled continuous-variable resource allocation and ILP offloading formulation; substantial complexity reduction.
[49]	2020	◇♣♡	ms & ec	Off	Matlab	Syn	ILP-based Optimization	Independent ILP formulations for caching and offloading then alternatively optimizes caching and offloading decisions.s
[119]	2020	◇	ec	Off	–	Syn	Convex Optimization	Requirement of full knowledge of task dependencies and system parameters, being impractical for continuous and dynamic environments.
[50]	2020	◇♡	ms	Off	–	Real-world Applications [120] + Google Trace [117]	Convex Optimization	A convex programming algorithm for general settings and a successor-based scheme tailored to homogeneous edge nodes with a competitive ratio of O(1).
[48]	2021	◇♡	ms	Off	–	Real-world Applications [120] + Google Trace [117]	Convex Optimization	Fully static caching assumption; limited applicability to dynamic environments.

Acronyms used in this table: Makespan(ms), Energy Consumption (ec), Cost(c), Offline(Off), Online(On). The use of "&" in Optimization Objective indicates joint optimization, while ";" denotes Pareto optimization. ◇ denotes Task Offloading, ♣ denotes Resource Allocation, ♡ denotes Service Caching, ♠ denotes Others.

A detailed review and in-depth analysis of the state-of-the-art studies corresponding to each of these four techniques are presented in Section 5.

5. Classification and review of workflow scheduling approaches

In this section, we grouped and reviewed existing studies in four categories: mathematical programming approaches, heuristic approaches, meta-heuristic approaches, and Deep Reinforcement Learning (DRL) approaches, as shown in Fig. 6. This classification is structured to navigate the readers through different technical roadmaps and to revisit the key elements of mainstream algorithm design.

5.1. Mathematical programming

Mathematical programming is a widely used tool for solving optimization problems, involving techniques such as linear programming, integer programming, nonlinear programming, and convex optimization. Mathematical programming not only defines NP-hard problems for edge workflow scheduling but also provides an effective solution for static, offline scheduling. It offers the key advantage of delivering provable performance guarantees at a controllable cost, which is crucial for practical application scenarios. In recent years, these methods have been extensively applied in the field of edge workflow scheduling.

Alsurdeh et al. [44] proposed a single workflow scheduling framework for edge cloud computing. Their two-stage approach first applies a Gradient Descent-based resource estimation algorithm to group tasks and determine the optimal number of cores. Then, a cluster-based scheduling algorithm allocates tasks across provisioned edge and cloud cores. However, there is no discussion of how to merge, interleave, or jointly optimize multiple independent workflows, limiting its applicability in dynamic, large-scale environments.

Yan et al. [45] jointly investigated task offloading and resource allocation in MEC systems with inter-user task dependencies, where a task's output on one user may serve as the input for a dependent task on another user. The authors formulated the problem as a mixed-integer nonlinear programming (MINLP) model, deriving closed-form solutions for the resource allocation problem, which is the only sub-problem involving continuous variables, via bisection search. They then translated the offloading problem into a linear programming (ILP) problem and proposed a Gibbs sampling-based method for solving it. Building on this work, Bi et al. [49] expanded the scope to include service caching alongside task offloading and resource allocation, optimizing for makespan and energy under broader constraints. The optimization problem is initially formulated as a MINLP model. Similarly, the authors first derive a closed-form solution for the resource allocation sub-problem, simplifying the original MINLP into a 0–1 ILP problem. To further reduce complexity, task offloading and service caching are divided into individual ILP problems, and an iterative alternating minimization approach is proposed to solve them efficiently.

Convex optimization is another widely used mathematical programming approach. In Mehrabi et al. [119], the authors formulated an energy-efficient offloading problem of dependent tasks in a three-node MEC system as a MINLP. They then recast this MINLP as a quadratic-constrained quadratic program and applied semidefinite relaxation to derive a convex semidefinite programming problem. Finally, feasible binary offloading decisions can be obtained through a randomized rounding procedure. Recognizing that tasks cannot be executed arbitrarily unless their required services are already cached, Zhao et al. [48] and their earlier work [50] defined a dependency-aware offloading problem with service caching, which is NP-hard. To address this, they proposed a convex programming-based algorithm for general scenarios and a favorite successor-based algorithm for cases with homogeneous edge nodes.

Despite their effectiveness, these works generally assume static knowledge of environmental information, e.g., channel conditions and task attributes, which limits their applicability in dynamic, unpredictable scenarios. For ease of understanding and comparison, Table 2 summarizes the information of the mathematical programming-based works introduced.

5.2. Heuristic approaches

Given the NP-hard nature of workflow scheduling within edge computing environments [36], it is typically infeasible to obtain optimal solutions in polynomial time. Heuristic algorithms are thus widely adopted, as they strike a practical balance between optimality and computational efficiency. This balance is crucial in edge computing, where resources are constrained compared to traditional cloud environments. Heuristics can provide near-optimal solutions in short periods, making them well suited for the dynamic and delay sensitive requirements of workflow scheduling in edge computing. Common heuristic methods, such as Greedy Algorithms, Auction Algorithms, and Local Search Algorithms, have been extensively applied in this context.

Many heuristic approaches for workflow scheduling are based on the list scheduling framework [19,82,91,121]. This well-established technique operates in two main phases: task prioritization and resource selection. In the first phase, all tasks in the workflow (DAG) are ranked according to specific criteria and placed in an ordered list. Classic schedulers like HEFT-based schedulers use a static upward rank [91]. More recent methods introduce advanced, resource-aware ranking metrics to generate more efficient task orders. For instance, Noorian Talouki et al. [121] proposed metrics based on Optimistic Cost Tables. Similarly, Asghari Alaie et al. [82] developed a new downward ranking policy that incorporates factors including in-degree total communication cost and the standard deviation of task execution times. The second phase, resource selection, involves iteratively mapping tasks from the ordered list to resources according to specific optimization techniques such as the greedy algorithm, auction algorithm, etc.

5.2.1. *Greedy algorithm*

Greedy algorithm is a low-complexity, well-adopted method that operates by seeking local optima at each step. It selects actions based on specific criteria, e.g., the shortest makespan or the lowest cost, to achieve immediate benefits.

Kanemitsu et al. [51] proposed a list scheduling-based algorithm, PCTSO, to address the dependent task offloading problem in a single-node edge computing environment. PCTSO first prioritizes tasks based on their remaining time to completion, ensuring that each task's priority exceeds that of its successors, thereby satisfying dependency constraints. For each task in the scheduling queue, PCTSO greedily decides whether to offload the task to the edge based on the expected makespan. Building on this work, Cai et al. [55] extended the approach to a multi-node edge environment, developing a similar greedy algorithm to minimize the makespan. To enhance resilience to potential edge server failures, the authors proposed a dependency-aware mechanism that quickly identifies and reschedules tasks impacted by server failures. In the context of a cloud-edge collaborative environment, Sahni et al. [57] proposed a heuristic algorithm for multi-hop offloading and network flow scheduling. This method treats each DAG as a set of co-subtask stages, assigning priorities and using a rank-based subtask list within each stage to make offloading and data transfer scheduling decisions. The heavy communication costs between remote clouds and edge servers are often underestimated, which limits the performance of the above work. DCDS, introduced by Lou et al. [74], tackles dependent task offloading in MEC-cloud environments with a dual-stage approach that separates edge and cloud scheduling. In the edge scheduling stage, tasks are assigned to edge servers using a greedy method to minimize cost while respecting the latest start time constraints of descendant tasks. If no edge server meets the constraints, cloud scheduling is activated. Specifically, an one-climb cloud strategy ensures that unschedulable tasks and their successors are fully offloaded to the cloud, avoiding frequent transmissions between the edge and cloud.

Previous research has primarily focused on offline scheduling approaches, which are less suitable for real-world scenarios characterized by frequent service requests and dynamic environmental conditions. In contrast, online workflow scheduling methods offer the flexibility to swiftly adapt to unforeseen tasks and environmental changes, enabling continuous adjustments to make optimal real-time decision-making. Liu et al. [58] developed an online offloading framework called COFE, which can monitor the context of the MEC-Cloud system in real-time to adaptively assign the dependent tasks to candidate computing devices. Based on COFE, a heuristic ranking-based algorithm was then proposed to assign tasks according to their bottom levels [122]. However, prior studies have often overlooked the runtime environment preparation required before task execution. Li et al. [78] addressed this gap by investigating dependent task offloading with on-demand function configuration in MEC environments to align tasks with suitable function environments. The authors initially developed an algorithm for single requests, later extending it to handle multiple randomly arriving requests named OnDoc. OnDoc prioritizes and constructs a scheduling queue across multiple workflows, then greedily selects an offloading server for each task to achieve the earliest finish time. Similarly, Lou et al. [59] introduced SDTS, which considers the startup latency of runtime environment preparation and employs a greedy strategy to select the edge server that minimizes task completion time. Uniquely, SDTS implements a cloud cloning mechanism, running a cloud-based replica alongside the edge task and retaining only the first completed instance, thereby enhancing resource efficiency and scheduling performance. In contrast to the above methods, which primarily apply greedy strategies for initial placement, Karami et al. [123] employed a greedy approach for fault tolerance and load balancing within their KCES online scheduling scheme. When a task fails due to insufficient resource allocation, horizontal roaming and vertical offloading are triggered, which greedily select the node with the largest residual resources for migration.

5.2.2. *Auction algorithm*

The Auction Algorithm is a distributed optimization technique where scheduling decisions are obtained through bidding processes. In this approach, participants bid on desired resources, and the algorithm iteratively adjusts resource prices based on demand, eventually achieving an equilibrium where each resource block is assigned to the highest bidder. Auction algorithms have been applied in edge computing scheduling, especially in scenarios where multiple users compete for limited edge resources.

To address resource competition among users, Liu et al. [75] proposed an auction-based offloading approach with a truthful bidding mechanism, encouraging participants to bid their true valuations to ensure fair selection. For each winning bidder, a heuristic offloading algorithm is applied. Additionally, the authors incorporated a Partial Critical Path(PCP) approach [124], which partitions the task graph into sequences, assigning each sequence to an individual edge device to reduce transfer time. Similarly, Hu et al. [66] introduced an auction-based scheme for online dependent task offloading in vehicular edge computing, where applications act as auctioneers and multiple RSUs as bidders. The RSU with the minimum energy consumption wins the bid. Additionally, they developed a server power management mechanism to further reduce the energy consumption of the overall system.

In addition to auction algorithms, there are also a few works, e.g., [125,126], on workflow scheduling in edge computing using other distributed techniques such as game theory and distributed consensus algorithms.

5.2.3. *Local search algorithm*

The Local Search Algorithm is an optimization method that iteratively explores neighboring solutions to improve the objective function. Starting with an initial solution, it moves to a neighboring solution if it offers improvement, repeating this process until no further gains can be made or a termination criterion is met.

Wang et al. [11] studied scheduling for mobile augmented reality applications with dependent tasks in a collaborative edge-cloud environment. The authors proposed Mutas, a scheduling algorithm that jointly addresses task offloading and resource allocation using a block coordinate descent approach, a typical local search technique. Specifically, Mutas can optimize task offloading or resource allocation decisions alternately while keeping the other term fixed; this process iteratively continues until either reaching a predefined maximum number of iterations or observing no significant improvement in the objective function. The last step in the Mutas algorithm is binary recovery where simulated annealing is incorporated to further improve the solution obtained from the block coordinate descent process.

For ease of understanding and comparison, Table 3 lists summary information about the introduced works based on heuristic approaches.

5.3. *Meta heuristic approaches*

While heuristic methods provide effective solutions for workflow scheduling within edge computing, their optimization performance is

Table 3
Review of heuristic-based works in edge workflow scheduling.

Paper	Year	Issue	Opt. objective	Sched. pattern	Simulation environment	Dataset	Approach	Highlights & limitations
[51]	2019	◇	ms	Off	CloudSim + WorkflowSim	Syn + Sci	Greedy	Typical sequential greedy scheduling framework; susceptibility to suboptimal decisions under complex dependencies.
[52]	2020	◇	ms	Off	Java	Syn + Sci	Greedy	Overlapping social-subnet modeling; enhanced environment representation of owner-based social communication patterns.
[85]	2020	◇	rel	Off	–	Syn	Greedy	Dependency-based constraint decomposition; reduced scheduling complexity.
[53]	2020	◇	ms	Off	–	Syn	Greedy	Strict reliance on idealized mobility and resource assumptions, being impractical for complex urban scenarios.
[11]	2020	◇♣	ms	On	–	Syn	Block Descending Algorithm	Online block-descending optimization; risk of slow convergence.
[55]	2021	◇ ♠	ms	Off	Python	Syn	Greedy	Failure-resilient DAG rescheduling; improved robustness under server failures and inherent resource bottlenecks.
[77]	2022	◇	sr	On	–	Syn	Greedy	Quantitative priority sorting across concurrent workflows; suitability for concurrent multi-workflow scheduling.
[56]	2021	◇♣	ms	Off	–	Syn	Greedy	DVFS-integrated offloading; flexible resource scaling on heterogeneous MEC servers
[57]	2021	◇♠	ms	Off	–	Syn	Greedy	Multi-hop offloading with network flow scheduling; better consideration of frequent communication overhead
[66]	2023	◇♠	ec	On	–	Syn	Auction	Dynamic power management via server sleep states; great energy efficiency without deadline violations.
[78]	2023	◇	sr	On	–	Alibaba Trace [118]	Greedy	Priority sorting across workflows in on-demand environment configuration; suitability for serverless context.
[58]	2021	◇	ms	On	CloudSim	Syn	Greedy	Real-time context monitoring and online task assignment.
[59]	2023	◇	ms	On	Python	Alibaba Trace [118]	Greedy	Cloud-cloning mechanism for makespan optimization; tradeoff with higher scheduling complexity.
[75]	2022	◇	profit	On	EdgeCloudSim + ElasticSim	Sci	Auction	Truthful auction offloading; fairness-guaranteed resource allocation
[74]	2023	◇	c	On	Python	Sci	Greedy	Consecutive task co-location; reduced frequent edge–cloud communication overhead.
[115]	2024	◇ ♣	resource utilization	On	Real-world testbed	Syn	Greedy	The proposed scheme, built upon a designed cloud-edge workflow engine, was validated using a real-world KubeEdge physical testbed.

Acronyms used in this table: makespan(ms), energy consumption (ec), cost(c), success rate(sr), reliability(rel), Offline(Off), Online(On), Synthetic Workflows(Syn), Scientific Workflows(Sci). The use of "&" in Optimization Objective indicates joint optimization, while ";" denotes Pareto optimization. ◇ denotes Task Offloading, ♣ denotes Resource Allocation, ♡ denotes Service Caching, ♠ denotes Others.

often limited by their tendency toward local gains. In contrast, meta-heuristic approaches emphasize a balance between exploration (global search) and exploitation (local search), enabling better searches in the solution space. Driven by this core benefit, meta-heuristic methods have been extensively used in the field of workflow scheduling [86,123,127]. Here we categorize and examine these works based on their use of meta-heuristic approaches, specifically Swarm Intelligence and Genetic Algorithms.

5.3.1. Swarm intelligence

Swarm Intelligence algorithms [128] are inspired by the collective behavior of decentralized, self-organized systems, such as flocks of birds, colonies of ants, or particle swarms. These algorithms leverage the principles of social interaction and cooperation to explore the solution space. The primary advantages of Swarm Intelligence include flexibility and the ability to approach near-optimal solutions through parallel processing and decentralized exploration. However, challenges such as premature convergence to sub-optimal solutions and the need for careful parameter tuning to balance exploration and exploitation effectively remain.

Xie et al. [41] proposed an enhanced Particle Swarm Optimization (PSO) algorithm to address the challenge of dependent task offloading in collaborative edge-cloud environments. In this model, each particle represents a potential solution, with each dimension corresponding to the offloading decision for individual workflow tasks. Following this system model, Zivkovic et al. [92] and Bacanin et al. [93] successively proposed two improved approaches. Zivkovic et al. [92] introduced an enhanced Harris Hawks Optimization algorithm, augmented with an opposition-based learning method. This approach generates opposite solutions for each hawk in every iteration, selecting the top half based on fitness values to form the new population, which improves solution diversity and prevents premature convergence, a common issue observed in the PSO algorithm used in Ref. [41]. Bacanin et al. [93] further improved the Firefly Algorithm (FA) by incorporating genetic operators, a quasi-reflection-based learning mechanism, and a dynamic noise parameter. These modifications strengthened the search capabilities of the original FA, facilitating a better balance between exploitation and exploration, and addressing the local optima problem in PSO.

Previous works often face challenges stemming from the large search space inherent in edge workflow scheduling problems, which increases computational complexity and hinders convergence for meta-heuristic algorithms. To address this, Kaur et al. [60] proposed a Bacterial Foraging Optimization Algorithm (BFOA)-based scheme that reduces the search space through a graph partitioning method. By clustering dependent tasks and treating them as single units during scheduling, the approach minimizes inter-task transmission delays and accelerates convergence. Additionally, a heuristic method is employed to improve the quality of the initial population, enhancing the BFOA's efficiency in cloud-edge environments. The goal of this work is to minimize the makespan.

Recognizing the uncertainties of edge environments, e.g., bandwidth variability, node failures, and workload fluctuations, various studies have proposed tailored workflow scheduling strategies. Peng et al. [86] introduced a novel reliability model in the MEC environment, which evaluates the connecting reliability of direct and multi-hop resource paths between nodes. They then proposed a Krill Herd-based algorithm to address the challenge of reliability-optimized workflow scheduling. Shao et al. [70] considered the scenario where workflows need to access multiple datasets in edge-cloud collaborative computing environments. The authors tackled data unavailability with a dynamic data replication framework across distributed datasets and an IṪO algorithm-based data replica scheduling approach. Works [70,86] model reliability with probability estimation, while Lin et al. [71] considered workflow execution as a fuzzy process and utilized triangular fuzzy numbers to model the lower and upper bound task execution and data transfer times. They then proposed an adaptive discrete PSO algorithm to seek offloading decisions with cost minimization. Considering workload fluctuations, Kuang et al. [47] integrated an artificial neural network to predict future workloads, enabling the MEC system to proactively allocate VMs for anticipated demand. They further proposed an enhanced Marine Predator Algorithm with opposition-based learning to minimize the workflow makespan, deadline violations, and the number of applied VMs.

5.3.2. *Genetic Algorithms*

Genetic Algorithms (GA) are a class of meta-heuristic algorithms inspired by the principles of natural selection, including operators such as selection, crossover, and mutation. Starting with a randomly generated population of potential solutions, GAs iteratively refine this population by prioritizing individuals with superior fitness, merging their attributes to produce offspring, and introducing mutations to maintain diversity within the solution space. Their primary strength lies in their robust global exploration capability, reducing the risk of local optima entrapment and improving the likelihood of identifying globally optimal solutions. However, the performance of GAs heavily depends on careful parameter design (e.g., population size, crossover, and mutation rates) and may require extensive iterations, posing challenges in delay sensitive scenarios.

In addressing task offloading for multiple DAG applications, Guo et al. [62] applied a GA-based algorithm to ultra-dense MEC systems for multi-workflow scheduling, having initially developed a heuristic algorithm for single-user scenarios. While these two studies primarily focus on single-objective optimization for makespan minimization, Pan et al. [106] proposed a Multi-objective Clustering Evolutionary Algorithm (MCEA) to address multi-workflow offloading within MEC environments. The algorithm aims to minimize both the cost and energy consumption of user devices while respecting hard deadline constraints. MCEA proactively filters out solutions that violate workflow deadlines during both population initialization and subsequent iterations and dynamically adjusts the probability of crossover and mutation to balance exploration and exploitation.

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is a widely used method to address the multi-objective optimization problem [133]. It offers a robust framework for achieving optimal solutions across various objectives. Peng et al. [103] employed NSGA-II to address the multi-objective workflow offloading problem in MEC, optimizing the makespan, cost, and energy consumption of user devices simultaneously. For resource allocation in robotic workflows within smart factories, Afrin et al. [35] proposed an improved NSGA-II algorithm, introducing a novel chromosome structure tailored to the resource allocation problem. Zhang et al. [46] focused on online workflow scheduling for scenarios such as anomaly detection and intelligent transportation, introducing a predictive NSGA-II algorithm. This algorithm leverages a Sequence-to-Sequence (Seq2Seq) model to generate high-quality initial solutions from historical data, significantly accelerating optimization and enabling online scheduling.

Building on the framework of NSGA-II, NSGA-III employs a reference point strategy to replace crowding distance, enhancing population diversity and reducing the risk of local optima [134]. Xu et al. [107] applied NSGA-III to dynamic resource provisioning within SDN-based edge computing frameworks, accounting for uncertainties in workflow execution. Their goal was to minimize both the makespan and energy consumption of the edge system simultaneously.

In summary, meta-heuristic approaches, including Swarm Intelligence and Genetic Algorithms, exhibit strong optimization capabilities and adaptability in addressing the complexities of edge workflow scheduling. However, challenges such as the need for precise parameter tuning, the risk of premature convergence, and slow convergence remain significant. Future research could explore the integration of advanced techniques, such as artificial intelligence, to enhance the efficiency of meta-heuristic methods, striving for faster convergence and improved optimization performance in complex scheduling scenarios. For ease of understanding and comparison, Table 4 lists summary information on introduced works based on meta-heuristic approaches.

5.4. *Deep reinforcement learning approaches*

Heuristic methods often provide suboptimal solutions, and meta-heuristic approaches can be computationally expensive, limiting their effectiveness in the dynamic, time-sensitive context of edge computing. DRL, which combines deep neural networks with reinforcement learning, offers a promising solution for rapid and effective scheduling in dynamic complex environments. This approach has gained significant attention for workflow scheduling in edge computing. To apply DRL effectively, it is essential to model the scheduling process as a Markov Decision Process (MDP), where decisions depend only on the current state, rather than past events. The MDP framework consists of several key components:

- **State:** This encapsulates the current conditions of the system, including the status of workflow tasks, the availability and load of edge resources, and the prevailing network conditions.
- **Action:** Represents potential scheduling decisions, such as task assignment choices, resource allocation strategies, or service placement.
- **Reward:** Measures the effectiveness of chosen actions according to key performance metrics such as workflow makespan, energy consumption, costs, etc.

DRL methods provide a robust framework for solving MDP. In a typical DRL framework, the agent interacts with the environment iteratively: observing the current state, selecting an action according to its policy, receiving a reward, and transitioning to a new state. The policy, represented by a neural network, is updated using either value-based methods, which estimate the expected rewards of state-action pairs, or policy-gradient methods, which optimize the action selection strategy directly.

This section reviews existing DRL-based workflow scheduling algorithms in the context of edge computing. Based on the characteristics of the DRL methods employed, related studies can be categorized into three main groups: value-based DRL methods, actor-critic DRL methods, and DRL methods incorporating graph neural networks.

5.4.1. *Value-based DRL*

Value-based DRL methods aim to estimate the expected return of actions in a given state, guiding the selection of optimal decisions [135]. A well-known example of this approach is the Deep Q-Network (DQN) and its variants, which represent a classic framework within value-based DRL.

Lu et al. [76] integrated an LSTM layer into the DQN framework to capture temporal dependencies in system states and mitigate overestimation issues. Similarly, Gao and Liu [79] proposed a Time-Improved DQN (TIDQN) that uses an LSTM layer to predict dynamic edge server load levels, enhancing adaptability to real-time changes. By adopting the Dueling DQN architecture, which separately estimates value and advantage functions, TIDQN improves training stability. The goal of this paper is to minimize deadline violations in workflows.

Several studies have investigated uncertainty in edge computing environments. Liu et al. [80] tackled the fluctuating performance of edge resources by incorporating probability mass functions derived from historical data for dynamic adaptation. They ultimately proposed a DQN-based algorithm to handle the offloading of dependent tasks with the goal of maximizing the probability that workflows meet the makespan constraints. Focusing on fault tolerance, Long et al. [63] addressed collaborative workflow scheduling in unreliable edge IoT environments. Specifically, considering hardware failures, they introduced a proactive fault tolerance mechanism using a Primary-Backup method

Table 4
Review of meta-heuristic-based works in edge workflow scheduling.

Paper	Year	Issue	Opt. objective	Sched. pattern	Simulation environment	Dataset	Approach	Highlights & limitations
[41]	2019	◇	ms & c	Off	WorkflowSim	Sci	PSO	A classical framework that represents each candidate workflow schedule as an individual.
[92]	2021	◇	ms & c	Off	WorkflowSim	Sci	Harris Hawks	Opposition-based diversity enhancement; increased algorithmic complexity.
[93]	2022	◇	ms & c	Off	WorkflowSim	Sci	Firefly	Noise-augmented firefly exploration; potential slow convergence
[60]	2022	◇	ms	Off	–	Syn	BFOA	This Graph-partitioning scheme and heuristic-based cluster initialization greatly reduce search-space.
[106]	2023	◇	ec; c	Off	Fog-WorkflowSim	Syn	GA	Proactive filtering of the deadline-violating individuals which accelerates convergence.
[103]	2019	◇	ms; ec; c	Off	Java	Syn	NSGA-II	Employment of classic NSGA-II framework, achieving multi-objective workflow scheduling.
[35]	2019	♣	ms; ec; c	Off	Matlab	Syn	NSGA-II	Novel chromosome structure tailored to the resource allocation problem.
[62]	2024	◇	ms	Off	–	Syn	GA	Extension to ultra dense edge network.
[86]	2019	◇	rel	Off	–	Sci + Real-world Trace [129]	Krill Herd	Particular investigation of the connecting reliability between nodes which are always overlooked.
[70]	2019	♠	c	On	Java	Sci	ITO	Dataset-centric optimization via dynamically replicating and scheduling distributed data, which could benefit ML-driven workflows.
[107]	2022	♠	ms; ec	Off	–	Syn	NSGA-III	NSGA-III for multi-objective workflow scheduling, trading higher complexity for improved solution performance.
[46]	2022	♣♠	c	On	WorkflowSim	Sci	NSGA-II	Neural network for predictive population initialization; reliance on historical data and prediction accuracy.
[47]	2022	♠	ms; sr	On	IFogSim	Syn + Real-world Applications [130–132]	Marine Predator	Workload-predictive VM provisioning, also being sensitive to prediction accuracy.
[67]	2022	◇	ec	Off	Matlab	Syn	GA	VM scaling based on neural networks; performance limited by prediction performance.
[71]	2021	◇	c	Off	Python	Sci	PSO	Fuzzy-bounds uncertainty modeling; improved scheduling robustness.

Acronyms used in this table: makespan(ms), energy consumption (ec), cost(c), success rate(sr), reliability(rel), Offline(Off), Online(On), Synthetic Workflows(Syn), Scientific Workflows(Sci). The use of "&" in Optimization Objective indicates joint optimization, while ";" denotes Pareto optimization. ◇ denotes Task Offloading, ♣ denotes Resource Allocation, ♡ denotes Service Caching, ♠ denotes Others.

and further developed a DQN-based algorithm to obtain offloading decisions, aimed at minimizing the makespan.

Previous works primarily focus on single-objective or joint optimization, as these approaches yield a single optimization value compatible with the reward calculation in the DQN framework. In contrast, Song et al. [108] tackled the multi-objective dependent task offloading problem in edge computing. The authors novelly formulated the problem as a multi-objective MDP with a vector-valued reward, where each element corresponds to an individual objective. To address this, they proposed a DQN-based algorithm designed to learn a Q-value vector rather than a scalar, enabling multi-objective optimization to simultaneously minimize the makespan, energy consumption of user devices, and cost.

5.4.2. *Actor-critic-based DRL*

Value-based DRL methods are effective for problems with discrete action spaces, but become impractical for scenarios requiring continuous action spaces, such as resource allocation in edge computing. Actor-critic-based DRL addresses this limitation by combining value estimation with policy optimization. This approach typically consists of an actor network that directly determines optimal actions through policy optimization, including continuous decisions like resource allocation, and a critic network that evaluates the effectiveness of these actions, providing feedback to refine the policy [142].

Yan et al. [96] addressed dependent task offloading and resource allocation in edge computing with an actor–critic framework. Their actor network combines an ordered retained action generation mechanism with Gaussian noise injection to promote exploration and accelerate policy learning, while their low-complexity critic network delivers rapid action evaluation. Considering the context of a partially observable edge environment, centralized optimization may be inefficient. Zhu et al. [64] thus proposed a distributed multi-agent actor-critic framework, employing decentralized actor networks and a centralized critic network to efficiently manage server allocation among multiple users, aimed at minimizing workflow makespan.

Several advanced DRL methods based on the actor-critic framework have been extensively used in this field, such as Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO). Liu et al. [81] studied workflow scheduling in a multi-slot system where applications arrive randomly, aiming to reduce deadline violations. The authors introduced a method to combine all active workflows into a unified DAG for each time slot. To achieve the optimal task execution sequence in the DAG, they designed a migration-enabled multi-priority task sequencing algorithm. Finally, a DDPG-based algorithm was presented to learn the optimal offloading policy. Zhao et al. [101] explored a dependent task offloading model in urban vehicular edge computing(VEC) and proposed a DDPG-based algorithm to train the offloading strategy. This approach integrates a mobility detection algorithm to improve training efficiency and a task priority scheme to enhance system stability.

Proximal Policy Optimization (PPO) is an advanced actor-critic DRL framework that improves training stability and efficiency by utilizing a clipped surrogate objective, which prevents excessively large policy updates, a common issue in traditional policy gradient methods. Li et al. [99] proposed a PPO-based algorithm for dependent task offloading, which simultaneously optimizes the makespan and energy consumption of user devices. This approach converts DAG-structured tasks into sequences using a novel graph sequence algorithm and employs a parameter-shared PPO network for efficient training. Expanding on multi-objective optimization, Tang et al. [109] applied a PPO-based algorithm in collaborative cloud-edge environments, focusing on Pareto optimization of makespan and energy consumption. They introduced an action mask mechanism into the PPO network to address the unavailability of offloading actions when IoT devices are outside the coverage of edge servers. Jayanetti et al. [68] further extended PPO for online workflow scheduling in cloud-edge environments, with a focus

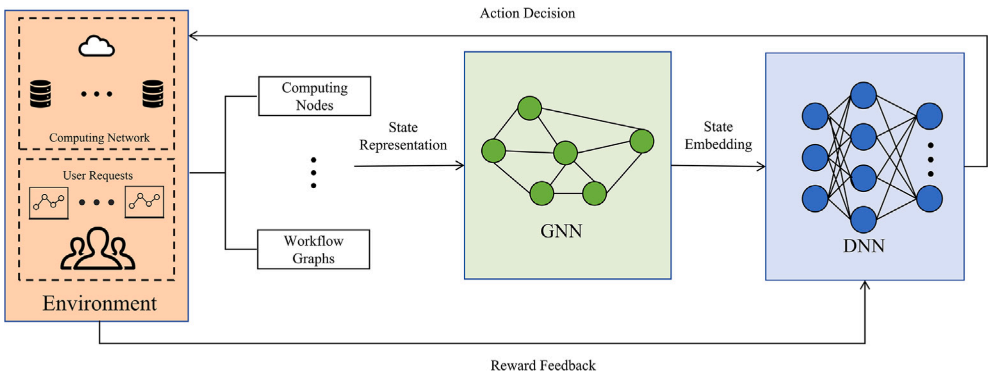

Fig. 7. GNN-enabled DRL architecture.

on system-wide energy minimization under workflow deadlines. Their MDP formulation introduces a hierarchical action space that explicitly separates edge-node and cloud-node decisions. The resulting actor-critic algorithm employs multiple actor networks alongside a single critic network, enabling efficient adaptation across heterogeneous execution layers.

5.4.3. *GNN-enabled DRL*

Traditional state representations in Deep Reinforcement Learning (DRL), particularly flattened vectors, are ill-equipped for workflow scheduling. Their primary limitation is the failure to perceive the complex, non-Euclidean structure of workflow DAGs. Consequently, they cannot leverage latent structural information and topological dependencies, treating them merely as constraints rather than as a rich source of information for optimal scheduling.

Graph Neural Networks (GNNs) directly address this representational deficiency. Through an aggregation process, GNNs excel at jointly capturing two critical components: the structural state, by encoding the complex non-Euclidean dependencies of the workflow DAG, and the environmental state, by modeling dynamic MEC resource characteristics, e.g., resource availability and node locality. This information is then formulated for the DRL agent, typically by either generating rich multi-level embeddings or fusing the workflow structural features and environmental features into a unified state representation. As illustrated in Fig. 7, the resulting representation provides a comprehensive topological understanding essential for formulating optimal policies. This approach has seen growing adoption, as the following literature demonstrates.

Tang et al. [72] introduced a graph convolutional network (GCN)-assisted DQN framework that improves dependency-aware task offloading by capturing complex task dependencies through GCNs which enhance state representations in DQN. Huang et al. [140] developed a GCN model to extract three types of embeddings including node embedding, workflow embedding, and global embedding. These multi-level embeddings provide a richer representation of task dependencies, individual workflow states, and global workflow loads respectively. Similarly, Ref. [110] proposed a GCN-based PPO framework with an intrinsic reward to guide agent training by estimating intermediate states during task dispatching. Despite this improvement, the framework primarily focuses on task dependencies without considering the dynamic nature of the MEC environment, limiting its effectiveness in real-time decision-making.

To address this, Mo et al. [138] introduced an approach that integrates MEC environment features into the state representation. Specifically, a GCN extracts task dependencies from the DAG, while a multilayer perceptron captures MEC-specific characteristics such as resource availability and device states. This unified state representation enables more comprehensive real-time decision-making for dependent task offloading, optimizing both workflow makespan and energy consumption. Similarly, Qin et al. [114] developed a GCN-based model to capture the features of the DAG and Kubernetes cluster environments respectively, followed by a DDQN-based algorithm to learn a scheduling policy that minimizes task completion time and balances resource utilization.

Cao et al. [139] utilized an advanced Graph Attention Network (GAT) for structural feature extraction among dependent tasks. Unlike prior works, this work pre-trained the GAT model in the cloud layer to handle the substantial computational requirements, subsequently utilizing the pre-trained model in the edge layer for real-time decision-making. This decoupling of training and execution significantly enhances the efficiency of the scheduling process, reducing computational overhead at the edge.

For ease of understanding and comparison, Table 5 summarizes the information on introduced DRL-based works.

5.5. *Critical synthesis and analysis*

To provide a critical synthesis beyond descriptive summaries, this section contrasts the primary algorithmic families along several cross-cutting dimensions. Table 6 presents this comparative analysis, which is then interpreted to distill trade-offs and actionable insights.

Interpreting this comparison reveals clear trade-offs. Mathematical programming methods, such as ILP, offer optimal or near-optimal solutions but suffer from severe scalability issues, making them impractical for large-scale, dynamic edge environments.

Heuristic methods are straightforward, efficient, and highly scalable, offering the fastest response times. However, they are typically static, perform poorly in dynamic environments, and often focus on local benefits, failing to find global optima.

Meta-heuristic methods, e.g., PSO, GA, NSGA-II, can search for the global optimum through iteration and generally achieve high-quality solutions for complex, multi-objective problems. However, this comes at the cost of low reproducibility (due to their stochastic nature) and high computational time, as they struggle with slow convergence. This makes them suitable for offline planning but ill-suited for online scheduling where quick responses are needed.

DRL methods are specifically characterized by strong adaptability and rapid decision-making, making them particularly suitable for the complex, dynamic environments found at the edge. However, this adaptability is earned through complex, data-hungry, and computationally expensive training phases. Furthermore, their real-world applicability and the overhead in resource-constrained edge environments require further validation.

This synthesis provides actionable insights for researchers and practitioners.

- For static, well-defined workflows where optimality is paramount and decisions can be made offline, meta-heuristic algorithms (e.g., GA, NSGA-II) typically offer the best performance.

Table 5
Review of DRL-based works in edge workflow scheduling.

Paper	Year	Issue	Opt. objective	Sched. pattern	Simulation environment	Dataset	Approach	Highlights & limitations
[76]	2020	◇	ec & c & lb	On	iFogSim	Syn	DQN	Temporal-dependency modeling via LSTM; overestimation mitigation
[108]	2022	◇	ms; ec; c	On	Python	Syn	DQN	Novel vector-reward MDP formulation to support Pareto optimization.
[79]	2022	◇	sr	On	Python	Syn	D3QN	LSTM-based load prediction, enhanced adaptability to real-time changes.
[36]	2023	◇	ms & ec & c	On	–	Syn	D3QN	Formal serverless-edge model with fully static service configuration, limited adaptability.
[80]	2020	◇	rel	On	–	Sci + Node position dataset [136]	DQN	Probability-based reliability modeling; real-time adaptation to resource fluctuations.
[63]	2022	◇♠	ms	On	–	Syn + Sci + Node position dataset [136]	DQN	Proactive fault tolerance via primary-backup, improved reliability at the cost of overhead.
[96]	2020	◇♣	ms & ec	On	Python	Syn	Actor-Critic	Low-complexity critic with Gaussian noise, which enhances exploration and fast evaluation.
[64]	2023	◇	ms	On	Python	Face Recognition Workflow [137]	MA Actor-Critic	Distributed actors with a central critic enable local decisions for each agent without inter-agent communication.
[109]	2023	◇	ms; ec	On	Python	Syn	PPO	Introduction of action-mask, which filters unavailable offloading choices.
[68]	2022	◇	ec	On	CloudSim	Syn	PPO	Hierarchical action space separates edge/cloud decisions for tailored policies among heterogeneous resource layers.
[99]	2022	◇	ms & ec	On	–	Real-world Applications [120]	PPO	Cost-based priority scoring, with inter-workflow priority relations omitted.
[100]	2022	◇	ms & ec	On	–	Syn	PPO	Seq2Seq-augmented state encoding, enriched representation with added overhead.
[81]	2023	◇	sr	On	Python	Syn	DDPG	Deadline-aware multi-priority sequencing, promoting deadline adherence under random arrivals.
[101]	2023	◇	ms & ec	On	Python	Syn	DDPG	Combination of mobility detection and task priority, contributed to policy training in the context vehicular edge computing.
[72]	2020	◇	c	On	–	Alibaba Trace [118]	GCN + DQN	GCN-based dependency embedding for better state representation; limited to single-workflow embedding.
[138]	2023	◇	ms & ec	On	–	Syn	GCN + DQN	MEC environment feature extraction via GCN and MLP, offering richer state at higher cost.
[139]	2024	◇	ms	On	Python	Syn	GAT + PPO	Pre-training of GAT in cloud, reduced edge overhead.
[140]	2024	◇	ms	On	Python	Alibaba Trace [118]	GCN + PG	Multi-level embedding extraction, stronger state modeling
[114]	2024	◇	ms	On	Real-world testbed	Sci	GCN + DDQN	GCN-based environment embedding, real Kubernetes cluster to simulate edge computing environment.
[110]	2024	◇	ms & ec	On	COSCO framework [141]	Syn + Sci	GCN + PPO	An intrinsic reward to guide agent training by estimating intermediate states during task dispatching.

Acronyms used: makespan (ms), energy consumption (ec), cost (c), load balance (lb), reliability (rel), success rate (sr), Offline(Off), Online(On), Deep Q-Network (DQN), Double Dueling DQN (DDQN), Double Dueling DQN (D3QN), Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), Graph Convolutional Network (GCN), Graph Attention Network (GAT), Synthetic Workflows(Syn), Scientific Workflows(Sci). The use of "&" indicates joint optimization of multiple objectives, while ";" denotes Pareto optimization. ◇ denotes Task Offloading, ♣ denotes Resource Allocation, ♡ denotes Service Caching, ♠ denotes Others.

Table 6
Critical comparison of workflow scheduling approach families.

Dimension	Mathematical Prog.	Heuristics	Meta-heuristics	DRL (Deep Reinforcement Learning)
Solution Optimality	High (Optimal/Approx.)	Moderate (Local Optima)	High (Near-Global Optima)	High(Learned Policy)
Scalability	Low	High	Moderate	High (Inference)/Low (Training)
Adaptability (to Variability)	Low (Static)	Low (Static)	Low (Must be re-run)	High
Response Time	High	Low	High (Slow convergence)	Low (Fast inference)
Reproducibility	High	High	Low (Stochastic)	Medium (Depends on training seed)
Implementation Complexity	Low	Low	High (Multiple iterations are needed)	High (High training overhead)

- For highly dynamic environments with unpredictable arrivals and changing resources, DRL-based approaches are preferable due to their high adaptability, assuming the significant training cost is feasible.
- For resource-constrained or real-time systems demanding immediate, low-overhead decisions, simple heuristics remain the most practical, despite their sub-optimal results.

6. Open challenges and future directions

This section aims to discuss some open challenges and outline potential future research directions in workflow scheduling within edge computing.

6.1. Fairness of workflow scheduling

As discussed in Section 5 on heuristic approaches, priority-based list scheduling is a widely adopted and computationally efficient framework for workflow management. However, a critical and still-open challenge, particularly in the dynamic and multi-tenant edge environment, is the inherent risk of task starvation. Starvation occurs when low-priority tasks are indefinitely postponed by a continuous influx of high-priority tasks, leading to unbounded waiting times and severe unfairness.

Therefore, a key future direction is the design of novel fairness-aware scheduling algorithms specifically for edge workflows. This research must go beyond simple starvation avoidance techniques, such as the aging mechanism [143], where a task's priority dynamically increases, and instead explore how to define and enforce different models of fairness (e.g., proportional fairness, max-min fairness) across intra/inter workflow and competing users. The central challenge lies in balancing this fairness with the stringent, low-latency demands of heterogeneous-priority edge applications.

6.2. Hierarchical data privacy and security in workflow scheduling

Data privacy and security are paramount concerns in edge computing, particularly within the context of workflow scheduling. Workflow scheduling inherently involves the distribution and execution of multiple interconnected tasks across diverse edge nodes, which often necessitates the sharing and processing of sensitive data. This distribution increases the risk of privacy breaches and data leaks, as private information generated on user devices may traverse various servers with differing security postures. Moreover, existing data privacy solutions in related work typically address isolated tasks without accounting for the complex dependencies and varying sensitivity levels inherent in workflow-based applications. For instance, in a smart healthcare workflow, patient identification data processed in one sub-task requires stringent privacy protections, whereas aggregated diagnostic data in another sub-task may permit more relaxed security measures. This disparity underscores the need for privacy mechanisms that can dynamically adapt to the discrete privacy requirements of each task of the workflow. Future research should focus on developing granular privacy protection strategies that align with the specific characteristics of workflow tasks. One promising approach involves hierarchical encryption techniques. By classifying data based on sensitivity at the source and applying appropriate encryption levels, it is possible to ensure that highly sensitive data is robustly protected without incurring unnecessary computational overhead for less sensitive information. Research in this field will facilitate the design of trustworthy and efficient workflow scheduling systems in edge computing, thereby fostering greater efficiency and trust in edge-based solutions for sensitive applications.

6.3. Topology-aware fault tolerance mechanism

In edge computing environments, potential task failures caused by hardware malfunctions, software errors, or network disruptions pose significant challenges to workflow scheduling. Furthermore, inherent dependencies within workflows mean that the failure of a single task can trigger a cascade of disruptions across multiple nodes, altering overall workflow execution and severely impacting workflows with hard deadlines [144,145], making the fault-tolerance mechanism an important research need.

Redundancy is a widely adopted fault-tolerance mechanism. Existing studies often employ either an application-centric approach, which deploys backups for entire workflow applications [63], or a task-centric approach, which introduces task redundancy based on individual task reliability requirements and failure probabilities [84]. However, these methods neglect the varying impacts of tasks on the global workflow. For example, the failure of tasks on the critical path can have severe consequences, whereas failures of tasks on non-critical paths may have relatively minor impacts. Consequently, these two approaches can introduce excessive redundancy for non-critical tasks, leading to unnecessary resource overhead. Addressing these challenges requires a shift towards more holistic fault-tolerance strategies that account for the complex dependencies and varying criticality of tasks within workflows. Future research should explore mechanisms that not only detect and mitigate individual task failures but also understand and manage their broader impact on the entire workflow.

6.4. Function-aware workflow scheduling in serverless computing

The serverless computing paradigm, particularly Function-as-a-Service (FaaS), is increasingly recognized as a key enabler for flexible and scalable application deployment at the network edge [146]. This paradigm shifts focus from managing hardware resources (CPU, memory) towards managing ephemeral function instances and their execution environments. Applications are decomposed into fine-grained, often stateless functions, requiring the underlying runtime to be available or instantiated on demand [59,78]. Consequently, tasks offloaded to edge servers can only execute once their required function environment is configured, introducing bottlenecks not typically considered in traditional, hardware-centric workflow scheduling. For example, invoking an image processing function necessitates the presence and readiness of the specific model and libraries on the target node.

Several studies [59,78] have started exploring function-aware scheduling, primarily treating function availability as an additional constraint. However, the ephemeral and dynamic nature of serverless execution necessitates deeper considerations, particularly concerning:

- **Cold Start Latency:** FaaS platforms often exhibit significant cold start delays due to the need for runtime initialization upon first invocation after inactivity. Within workflows, where functions execute interdependently, cumulative cold starts along the critical path can severely impact end-to-end latency. Mitigating these delays requires sophisticated strategies like predictive pre-warming, snapshotting, or intelligent placement anticipating spatio-temporal demand fluctuations, alongside proactive caching informed by function popularity across diverse workflows.
- **Function Chaining and Composition:** Serverless workflows frequently manifest as chains or DAGs of interconnected functions. Efficient scheduling must optimize not only individual function placement but also the inter-function data passing mechanisms (e.g., via edge storage or messaging queues) to minimize overhead and latency across the entire chain.
- **Resource Granularity and Management:** FaaS introduces a much finer resource granularity compared to VMs or containers. Scheduling involves managing numerous short-lived function instances across heterogeneous edge nodes, demanding lightweight isolation techniques and efficient resource multiplexing to ensure performance predictability and efficient utilization at scale.
- **Cost-Performance Trade-offs:** Serverless pricing models (e.g., pay-per-invocation) introduce new cost-performance trade-offs. Scheduling decisions must navigate the balance between reducing latency, potentially by over-provisioning or reducing reuse, and minimizing execution costs, often under user-defined budget constraints.

Furthermore, beyond these specific runtime challenges, the underlying deployment of functions onto the edge-fog-cloud infrastructure is inherently a critical optimization problem. This requires considering broader Quality-of-Service (QoS) metrics, such as energy models for sustainability and reliability models for resilience, which are critical for many IoT workflows. However, these objectives often conflict. As discussed in Ref. [147], optimizing for user-centric QoS goals, such as low latency, often creates a conflict between stakeholders. For instance, end-user objectives may clash with the resource providers' goal of minimizing operational costs. This conflict has been largely neglected in schedulers that focus purely on FaaS runtime optimization.

Addressing both the runtime-specific challenges, such as cold starts, and the strategic conflicts in deployment necessitates a shift towards co-designing workflow scheduling algorithms with function lifecycle management, intelligent caching, and network-aware data orchestration tailored specifically for the serverless edge environment.

6.5. Workflow scheduling over heterogeneous edge AI infrastructure

While substantial research on edge computing scheduling has traditionally focused on general CPU-centric resource models, modern AI-driven applications increasingly demand heterogeneous computing resources, including GPUs, NPUs, and specialized accelerators. These requirements stem from the widespread adoption of computationally intensive applications, including machine learning model training and complex data inference, which cannot be effectively supported by CPUs alone. However, current edge scheduling strategies typically treat computing resources as uniform CPU units, neglecting the unique performance profiles and constraints of diverse accelerators, resulting in the underutilization of valuable hardware resources. As edge computing scales to handle more intricate and data-intensive AI workflows, overlooking heterogeneous resource demands severely limits its potential benefits and practical applicability in real-world deployments.

In the context of workflow scheduling for Edge AI infrastructure, heterogeneous resource demands introduce additional complexity. A single workflow may encompass multiple stages, each with distinct performance requirements and hardware affinities. Consider a video analytics workflow designed for intelligent transportation systems. In its early stage, the workflow may involve sampling and basic frame-level preprocessing of incoming video streams, which can efficiently run on general-purpose CPUs. Subsequent tasks may involve computationally heavy operations, such as periodic model training and object tracking, which can greatly benefit from GPU or NPU accelerators with their high-throughput parallel architectures. Since the workflow can be distributed across a geographically scattered set of edge nodes, identifying nodes that host the appropriate accelerators and allocating the correct mix of hardware to each task segment are critical to optimizing the overall workflow performance.

Consequently, future research should incorporate heterogeneous resource modeling on edge AI infrastructure into the core of edge workflow scheduling strategies. This involves accurately characterizing the performance profiles of various AI infrastructures, developing benchmarks that reflect hardware capabilities within edge environments, and incorporating these insights into scheduling algorithms.

6.6. *Integrating edge computing with workflow-level scheduling in LLM services*

Large language models (LLMs) have garnered widespread attention and adoption owing to their exceptional capabilities in natural language understanding and generation. Current work has shown that harnessing the intrinsic parallelism of LLM services represents a promising avenue for further latency benefit. For instance, LLMCompiler [148] demonstrates how an incoming user query can be automatically broken down into a small directed graph of function-calling subtasks, which are then dispatched and executed in parallel, rather than one by one, yielding up to a several-fold reduction in end-to-end response time compared to purely sequential methods. Meanwhile, modeling LLM workflows at the level of individual primitives offers a distinct approach to workflow decomposition and scheduling. In Teola [149], queries are transformed into fine-grained workflows where each node represents a task primitives—such as token embedding, partial model prefill, or output decoding—and edges capture precise data dependencies. The scheduler then optimizes this primitive graph by eliminating redundant paths, segmenting heavy operations into pipeline stages, and applying cross-primitive batching to maximize concurrency. Compared to coarser orchestration, this primitive-aware strategy delivers up to a 2.09× reduction in end-to-end latency on real inference workloads. Notably, even simple heuristic schedulers, e.g., greedy dispatch in LLMCompiler and the depth-aware batching in Teola, yield multi-fold end-to-end latency reductions, underscoring the high potential of workflow-level scheduling for LLM workloads.

However, these methods generally remain tied to centralized environemnt and do not address the network latency and privacy limitations of cloud-centric architectures. Offloading LLM workflows to fully distributed edge nodes can dramatically shorten round-trip times, mitigate backbone congestion, and keep sensitive data local. Moreover, the edge's inherently dispersed fabric naturally complements workflow decomposition—enabling fine-grained task placement, cross-node parallelism, and locality-aware scheduling that centralized systems cannot match. Despite these clear benefits, the fusion of edge computing and workflow-level LLM scheduling is still in its infancy and deserves deeper exploration.

7. Conclusion and future directions

In conclusion, this survey systematically reviews workflow scheduling in edge computing with an in-depth exposition of the challenges unique to this evolving field. First, we present the motivation followed by a discussion of core challenges, thus elaborating on two critical questions that why edge workflow scheduling is necessary and why numerous cloud-based workflow scheduling strategies are not directly applicable to edge computing. Then we introduce the background knowledge of this field including the representative application scenarios and formalized basic model. An important contribution of this survey is the multidimensional taxonomy of existing workflow scheduling strategies categorized by research issues, optimization objectives, scheduling patterns, and optimization approaches, as well as the simulation environments and datasets used for evaluation. Moreover, the survey discusses open challenges and potential future research directions that cover hierarchical data privacy and security, topology-aware fault tolerance, function-aware workflow scheduling in serverless computing, and the exploration of heterogeneous computing resources in edge workflow scheduling and workflow-level scheduling for emerging LLM services. We acknowledge that this review, while comprehensive, is inherently limited by the established search scope and the rapid publication velocity within this domain. Nevertheless, through an comprehensive investigation of the current state of the art and the discussion of research gaps and future directions, we believe this survey can offer researchers a holistic and insightful viewpoint, and contribute to more effective and robust workflow scheduling strategies within the rapidly evolving edge computing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by National Natural Science Foundation of China (62402198), Fundamental Research Funds for the Central Universities (21624348), Guangdong Provincial Natural Science Foundation Project (2025A1515010113), Shandong Provincial Natural Science Foundation Project (ZR2024LZH012), and the Major Key Project of PCL, China under Grant PCL2025A08 and PCL2025A11.

Data availability

No data was used for the research described in the article.

References

[1] S. Sukhmani, M. Sadeghi, M. Erol-Kantarci, A. El Saddik, Edge caching and computing in 5G for mobile AR/VR and tactile internet, IEEE MultiMed. 26 (1) (2019) 21–30, https://doi.org/10.1109/MMUL.2018.2879591
[2] K. Zheng, L. Hou, H. Meng, Q. Zheng, N. Lu, L. Lei, Soft-defined heterogeneous vehicular network: architecture and challenges, IEEE Netw. 30 (4) (2016) 72–80, https://doi.org/10.1109/MNET.2016.7513867
[3] M. Pouryazdan, B. Kantarci, T. Soyata, H. Song, Anchor-assisted and vote-based trustworthiness assurance in smart city crowdsensing, IEEE Access 4 (2016) 529–541, https://doi.org/10.1109/ACCESS.2016.2519820
[4] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: the communication perspective, IEEE Commun. Surv. & Tutorials 19 (4) (2017) 2322–2358, https://doi.org/10.1109/COMST.2017.2745201
[5] J. Li, W. Liang, W. Xu, Z. Xu, X. Jia, A.Y. Zomaya, S. Guo, Budget-aware user satisfaction maximization on service provisioning in mobile edge computing, IEEE Trans. Mob. Comput. 22 (12) (2023) 7057–7069, https://doi.org/10.1109/TMC.2022.3205427
[6] J. Ren, G. Yu, Y. He, G.Y. Li, Collaborative cloud and edge computing for latency minimization, IEEE Trans. Veh. Technol. 68 (5) (2019) 5031–5044, https://doi.org/10.1109/TVT.2019.2904244

[7] H. Lin, S. Zeadally, Z. Chen, H. Labiod, L. Wang, A survey on computation offloading modeling for edge computing, J. Netw. Comput. Appl. 169 (2020) 102781.
[8] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81, https://doi.org/10.1109/MC.2016.145
[9] X. Ma, A. Zhou, S. Zhang, S. Wang, Cooperative service caching and workload scheduling in mobile edge computing, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 2076–2085, https://doi.org/10.1109/INFOCOM41043.2020.9155455
[10] H. Chen, H. Qin, W. Chen, N. Li, T. Wang, J. He, G. Yang, Y. Peng, BMS: bandwidth-aware multi-interface scheduling for energy-efficient and delay-constrained gateway-to-device communications in IOT, Comput. Netw. 225 (2023) 109645. ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2023.109645, https://www.sciencedirect.com/science/article/pii/S1389128623000907.
[11] C. Wang, S. Zhang, Z. Qian, M. Xiao, J. Wu, B. Ye, S. Lu, Joint server assignment and resource management for edge-based MAR system, IEEE/ACM Trans. Netw. 28 (5) (2020) 2378–2391, https://doi.org/10.1109/TNET.2020.3012410
[12] M.S. Elbamby, C. Perfecto, C.-F. Liu, J. Park, S. Samarakoon, X. Chen, M. Bennis, Wireless edge computing with latency and reliability guarantees, Proc. IEEE 107 (8) (2019) 1717–1737, https://doi.org/10.1109/JPROC.2019.2917084
[13] E. Bozkaya, Digital twin-assisted and mobility-aware service migration in mobile edge computing, Comput. Netw. 231 (2023) 109798. ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2023.109798, https://www.sciencedirect.com/science/article/pii/S1389128623002438.
[14] C.-H. Hong, B. Varghese, Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms, ACM Comput. Surv. 52 (5) (Sep 2019). ISSN 0360-0300, https://doi.org/10.1145/3326066
[15] Q. Luo, S. Hu, C. Li, G. Li, W. Shi, Resource scheduling in edge computing: a survey, IEEE Commun. Surv. Tutor. 23 (4) (2021) 2131–2165, https://doi.org/10.1109/COMST.2021.3106401
[16] Y. Sahni, J. Cao, L. Yang, S. Wang, Distributed resource scheduling in edge computing: problems, solutions, and opportunities, Comput. Netw. 219 (2022) 109430. ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2022.109430, https://www.sciencedirect.com/science/article/pii/S1389128622004649.
[17] X. Xia, S.M.M. Fattah, M.A. Babar, A survey on UAV-enabled edge computing: resource management perspective, ACM Comput. Surv. 56 (3) (Oct 2023). ISSN 0360-0300, https://doi.org/10.1145/3626566
[18] X. Zhang, S. Debroy, Resource management in mobile edge computing: a comprehensive survey, ACM Comput. Surv. 55 (13s) (Jul 2023). ISSN 0360-0300, https://doi.org/10.1145/3589639
[19] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, J. Supercomput. 71 (9) (May 2015) 3373–3418. ISSN 1573-0484, https://doi.org/10.1007/s11227-015-1438-4, http://dx.doi.org/10.1007/s11227-015-1438-4.
[20] M. Adhikari, T. Amgoth, S.N. Srirama, A survey on scheduling strategies for workflows in cloud environment and emerging trends, ACM Comput. Surv. 52 (4) (Aug 2019). ISSN 0360-0300, https://doi.org/10.1145/3325097
[21] M. Hosseinzadeh, M.Y. Ghafour, H.K. Hama, B. Vo, A. Khoshnevis, Multi-objective task and workflow scheduling approaches in cloud computing: a comprehensive review, J. Grid Comput. 18 (3) (Sep 2020) 327–356. ISSN 1572-9184, https://doi.org/10.1007/s10723-020-09533-z, http://dx.doi.org/10.1007/s10723-020-09533-z.
[22] R. Bouabdallah, F. Fakhfakh, Workflow scheduling in cloud–fog computing environments: a systematic literature review, Concurr. Comput. Pract. Exp. 36 (28) (2024) e8304.
[23] N. Khaledian, M. Voelp, S. Azizi, M.H. Shirvani, AI-based & heuristic workflow scheduling in cloud and fog computing: a systematic review, Clust. Comput. 27 (8) (2024) 10265–10298.
[24] A. Jayanetti, S. Halgamuge, R. Buyya, Reinforcement learning based workflow scheduling in cloud and edge computing environments: A taxonomy, review and future directions, arXiv preprint arXiv:2408.02938, 2024.
[25] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 2014, pp. 1–10.
[26] A. Nunes, K.W. Axhausen, Road safety, health inequity and the imminence of autonomous vehicles, Nat. Mach. Intell. 3 (8) (2021) 654–655.
[27] J. Zhao, W. Zhao, B. Deng, Z. Wang, F. Zhang, W. Zheng, W. Cao, J. Nan, Y. Lian, A.F. Burke, Autonomous driving system: a comprehensive survey, Expert Syst. Appl. (2023) 122836.
[28] B. Badjie, J. Cecilio, A. Casimiro, Adversarial attacks and countermeasures on image classification-based deep learning models in autonomous driving systems: a systematic review, ACM Comput. Surv. 57 (1) (2024) 1–52.
[29] D.-H. Lee, J.-L. Liu, End-to-end deep learning of lane detection and path prediction for real-time autonomous driving, Signal Image Video Process. 17 (1) (2023) 199–205.
[30] Y. Fu, C. Li, F.R. Yu, T.H. Luan, P. Zhao, An incentive mechanism of incorporating supervision game for federated learning in autonomous driving, IEEE Trans. Intell. Transp. Syst. 24 (12) (2023) 14800–14812.
[31] W.-B. Kou, Q. Lin, M. Tang, S. Xu, R. Ye, Y. Leng, S. Wang, G. Li, Z. Chen, G. Zhu, et al., pFedLVM: a large vision model (LVM)-driven and latent feature-based personalized federated learning framework in autonomous driving, IEEE Trans. Intell. Transp. Syst. 26 (10) (2025) 15915–15931.
[32] K. Lin, B. Lin, X. Chen, Y. Lu, Z. Huang, Y. Mo, A time-driven workflow scheduling strategy for reasoning tasks of autonomous driving in edge environment, in: 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), 2019, pp. 124–131, https://doi.org/10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00028
[33] Y. Li, C. Yang, X. Chen, Y. Liu, Mobility and dependency-aware task offloading for intelligent assisted driving in vehicular edge computing networks, Vehic. Commun. 45 (2024) 100720.
[34] M. Aazam, S. Zeadally, K.A. Harras, Deploying fog computing in industrial internet of things and industry 4.0, IEEE Trans. Ind. Inf. 14 (10) (2018) 4674–4682.
[35] M. Afrin, J. Jin, A. Rahman, Y.-C. Tian, A. Kulkarni, Multi-objective resource allocation for edge cloud based robotic workflow in smart factory, Futur. Gener. Comput. Syst. 97 (2019) 119–130.
[36] R. Xie, D. Gu, Q. Tang, T. Huang, F.R. Yu, Workflow scheduling in serverless edge computing for the industrial internet of things: a learning approach, IEEE Trans. Ind. Inf. 19 (7) (2023) 8242–8252, https://doi.org/10.1109/TII.2022.3217477
[37] B.-S. Sun, H. Huang, Z.-Y. Chai, Y.-J. Zhao, H.-S. Kang, Multi-objective optimization algorithm for multi-workflow computation offloading in resource-limited IIoT, Swarm Evol. Comput. 89 (2024) 101646.
[38] X. Jiang, F.R. Yu, T. Song, V.C.M. Leung, A survey on multi-access edge computing applied to video streaming: some research issues and challenges, IEEE Commun. Surv. & Tutorials 23 (2) (2021) 871–903, https://doi.org/10.1109/COMST.2021.3065237
[39] W. Shi, Q. Li, Q. Yu, F. Wang, G. Shen, Y. Jiang, Y. Xu, L. Ma, G.-M. Muntean, A survey on intelligent solutions for increased video delivery quality in cloud-edge-end networks, IEEE Commun. Surv. Tutor. (2024) 1, https://doi.org/10.1109/COMST.2024.3427360
[40] M.A. Khan, E. Baccour, Z. Chkirbene, A. Erbad, R. Hamila, M. Hamdi, M. Gabbouj, A survey on mobile edge computing for video streaming: opportunities and challenges, IEEE Access 10 (2022) 120514–120550, https://doi.org/10.1109/ACCESS.2022.3220694
[41] Y. Xie, Y. Zhu, Y. Wang, Y. Cheng, R. Xu, A.S. Sani, D. Yuan, Y. Yang, A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment, Futur. Gener. Comput. Syst. 97 (2019) 361–378.
[42] C. Rong, J.H. Wang, J. Liu, J. Wang, F. Li, X. Huang, Scheduling massive camera streams to optimize large-scale live video analytics, IEEE/ACM Trans. Netw. 30 (2) (2022) 867–880, https://doi.org/10.1109/TNET.2021.3125359
[43] J. Jin, J. Gubbi, S. Marusic, M. Palaniswami, An information framework for creating a smart city through internet of things, IEEE Internet Things J. 1 (2) (2014) 112–121, https://doi.org/10.1109/JIOT.2013.2296516
[44] R. Alsurdeh, R.N. Calheiros, K.M. Matawie, B. Javadi, Hybrid workflow provisioning and scheduling on edge cloud computing using a gradient descent search approach, in: 2020 19th International Symposium on Parallel and Distributed Computing (ISPDC), 2020, pp. 68–75, https://doi.org/10.1109/ISPDC51135.2020.00019
[45] J. Yan, S. Bi, Y.J. Zhang, M. Tao, Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency, IEEE Trans. Wireless Commun. 19 (1) (2020) 235–250, https://doi.org/10.1109/TWC.2019.2943563
[46] M. Zhang, Z. Yang, J. Yan, S. Ali, W. Ding, G. Wang, Task-load aware and predictive-based workflow scheduling in cloud-edge collaborative environment, J. Rel. Intell. Environ. 8 (1) (2022) 35–47.
[47] F. Kuang, Z. Xu, M. Masdari, Multi-workflow scheduling and resource provisioning in mobile edge computing using opposition-based marine-predator algorithm, Pervasive Mob. Comput. 87 (2022) 101715.
[48] G. Zhao, H. Xu, Y. Zhao, C. Qiao, L. Huang, Offloading tasks with dependency and service caching in mobile edge computing, IEEE Trans. Parallel Distrib. Syst. 32 (11) (2021) 2777–2792, https://doi.org/10.1109/TPDS.2021.3076687
[49] S. Bi, L. Huang, Y.-J.A. Zhang, Joint optimization of service caching placement and computation offloading in mobile edge computing systems, IEEE Trans. Wireless Commun. 19 (7) (2020) 4947–4963, https://doi.org/10.1109/TWC.2020.2988386
[50] G. Zhao, H. Xu, Y. Zhao, C. Qiao, L. Huang, Offloading dependent tasks in mobile edge computing with service caching, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, 2020, pp. 1997–2006, https://doi.org/10.1109/INFOCOM41043.2020.9155396
[51] H. Kanemitsu, M. Hanada, H. Nakazato, Multiple Workflow Scheduling with Offloading Tasks to Edge Cloud, Springer, 2019, pp. 38–52. ISBN 9783030235024, https://doi.org/10.1007/978-3-030-23502-4_4.
[52] J. Sun, L. Yin, M. Zou, Y. Zhang, T. Zhang, J. Zhou, Makespan-minimization workflow scheduling for complex networks with social groups in edge computing, J. Syst. Archit. 108 (2020) 101799 ISSN 1383-7621, https://doi.org/10.1016/j.sysarc.2020.101799, https://www.sciencedirect.com/science/article/pii/S1383762120300928.
[53] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, F. Yang, Dependency-aware task scheduling in vehicular edge computing, IEEE Internet Things J. 7 (6) (2020) 4961–4971, https://doi.org/10.1109/JIOT.2020.2972041
[54] J. Lee, H. Ko, J. Kim, S. Pack, Data: dependency-aware task allocation scheme in distributed edge clouds, IEEE Trans. Ind. Inf. 16 (12) (2020) 7782–7790, https://doi.org/10.1109/TII.2020.2990674
[55] L. Cai, X. Wei, C. Xing, X. Zou, G. Zhang, X. Wang, Failure-resilient DAG task scheduling in edge computing, Comput. Netw. 198 (2021) 108361.
[56] J. Liang, K. Li, C. Liu, K. Li, Joint offloading and scheduling decisions for DAG applications in mobile edge computing, Neurocomputing 424 (2021) 160–171.
[57] Y. Sahni, J. Cao, L. Yang, Y. Ji, Multihop offloading of multiple DAG tasks in collaborative edge computing, IEEE Internet Things J. 8 (6) (2021) 4893–4905, https://doi.org/10.1109/JIOT.2020.3030926

[58] J. Liu, J. Ren, Y. Zhang, X. Peng, Y. Zhang, Y. Yang, Efficient dependent task offloading for multiple applications in MEC-cloud system, IEEE Trans. Mob. Comput. (2021).
[59] J. Lou, Z. Tang, W. Jia, W. Zhao, J. Li, Startup-aware dependent task scheduling with bandwidth constraints in edge computing, IEEE Trans. Mob. Comput. (2023).
[60] M. Kaur, S. Kadam, N. Hannoon, Multi-level parallel scheduling of dependent-tasks using graph-partitioning and hybrid approaches over edge-cloud, Soft Comput. 26 (11) (2022) 5347–5362.
[61] Q. Li, B. Peng, Q. Li, M. Lin, C. Chen, S. Peng, A latency-optimal task offloading scheme using genetic algorithm for DAG applications in edge computing, in: 2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), 2023, pp. 344–348, https://doi.org/10.1109/ICCCBDA56900.2023.10154698
[62] M. Guo, X. Hu, Y. Chen, Y. Yang, L. Zhang, L. Chen, Joint scheduling and offloading schemes for multiple interdependent computation tasks in mobile edge computing, IEEE Internet Things J. 11 (4) (2024) 5718–5730, https://doi.org/10.1109/JIOT.2023.3307769
[63] T. Long, Y. Ma, L. Wu, Y. Xia, N. Jiang, J. Li, X. Fu, X. You, B. Zhang, A novel fault-tolerant scheduling approach for collaborative workflows in an edge-IOT environment, Digit. Commun. Netw. 8 (6) (2022) 911–922.
[64] K. Zhu, Z. Zhang, F. Sun, Toward intelligent cooperation at the edge: improving the QOS of workflow scheduling with the competitive cooperation of edge servers, Wireless Netw. (2023) 1–13.
[65] K. Zhu, Z. Zhang, F. Sun, B. Shen, Workflow makespan minimization for partially connected edge network: a deep reinforcement learning-based approach, IEEE Open J. Commun. Soc. 3 (2022) 518–529, https://doi.org/10.1109/OJCOMS.2022.3158417
[66] B. Hu, Y. Shi, Z. Cao, Adaptive energy-minimized scheduling of real-time applications in vehicular edge computing, IEEE Trans. Ind. Inf. 19 (5) (2023) 6895–6906, https://doi.org/10.1109/TII.2022.3207754.
[67] S. Chakraborty, K. Mazumdar, Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing, J. King Saud Univ. Comput. Inf. Sci. 34 (4) (2022) 1552–1568.
[68] A. Jayanetti, S. Halgamuge, R. Buyya, Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments, Futur. Gener. Comput. Syst. 137 (2022) 14–30.
[69] R. Salimi, S. Azizi, J. Abawajy, A greedy randomized adaptive search procedure for scheduling IOT tasks in virtualized fog–cloud computing, Trans. Emerg. Telecommun. Technol. 35 (5) (2024) e4980.
[70] Y. Shao, C. Li, Z. Fu, L. Jia, Y. Luo, Cost-effective replication management and scheduling in edge computing, J. Netw. Comput. Appl. 129 (2019) 46–61.
[71] B. Lin, C. Lin, X. Chen, A cost-driven fuzzy scheduling strategy for intelligent workflow decision making systems in uncertain edge-cloud environments, arXiv preprint arXiv:2107.01405, 2021.
[72] Z. Tang, J. Lou, F. Zhang, W. Jia, Dependent task offloading for multiple jobs in edge computing, in: 2020 29th International Conference on Computer Communications and Networks (ICCCN), 2020, pp. 1–9, https://doi.org/10.1109/ICCCN49398.2020.9209593
[73] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, X. Wang, Multitask offloading strategy optimization based on directed acyclic graphs for edge computing, IEEE Internet Things J. 9 (12) (2021) 9367–9378.
[74] J. Lou, Z. Tang, S. Zhang, W. Jia, W. Zhao, J. Li, Cost-effective scheduling for dependent tasks with tight deadline constraints in mobile edge computing, IEEE Trans. Mob. Comput. (2022).
[75] J. Liu, Y. Zhang, J. Ren, Y. Zhang, Auction-based dependent task offloading for IOT users in edge clouds, IEEE Internet Things J. 10 (6) (2022) 4907–4921.
[76] H. Lu, C. Gu, F. Luo, W. Ding, X. Liu, Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning, Futur. Gener. Comput. Syst. 102 (2020) 847–861.
[77] H. Liao, X. Li, D. Guo, W. Kang, J. Li, Dependency-aware application assigning and scheduling in edge computing, IEEE Internet Things J. 9 (6) (2022) 4451–4463, https://doi.org/10.1109/JIOT.2021.3104015
[78] G. Li, H. Tan, L. Liu, H. Zhou, S.H.-C. Jiang, Z. Han, X.-Y. Li, G. Chen, DAG scheduling in mobile edge computing, ACM Trans. Sen. Netw. 20 (1) (Oct 2023). ISSN 1550-4859, https://doi.org/10.1145/3616374
[79] Y. Gao, W. Liu, Multiple workflows offloading based on improved deep Q-network in mobile edge computing, in: 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2022, pp. 1281–1286, https://doi.org/10.1109/CSCWD54268.2022.9776251
[80] H. Liu, Y. Ma, P. Chen, Y. Xia, Y. Ma, W. Zheng, X. Li, Scheduling multi-workflows over edge computing resources with time-varying performance, a novel probability-mass function and DQN-based approach, in: Web Services–ICWS 2020: 27th International Conference, Held as Part of the Services Conference Federation, SCF 2020, Honolulu, HI, USA, September 18–20, 2020, Proceedings 27, Springer, 2020, pp. 197–209.
[81] S. Liu, Y. Yu, X. Lian, Y. Feng, C. She, P.L. Yeoh, L. Guo, B. Vucetic, Y. Li, Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks, IEEE J. Sel. Areas Commun. 41 (2) (2023) 538–554, https://doi.org/10.1109/JSAC.2022.3233532
[82] Y. Asghari Alaie, M. Hosseini Shirvani, A.M. Rahmani, A hybrid bi-objective scheduling algorithm for execution of scientific workflows on cloud platforms with execution time and reliability approach, J. Supercomput. 79 (2) (2023) 1451–1503.
[83] K. Li, Integrated analysis of reliability, power, and performance for IOT devices and servers, J. Syst. Archit. 154 (2024) 103216.
[84] R. Liu, W. Wu, X. Guo, G. Zeng, K. Li, Replica fault-tolerant scheduling with time guarantee under energy constraint in fog computing, Futur. Gener. Comput. Syst. 159 (2024) 567–579.
[85] Y. Shang, J. Li, X. Wu, DAG-based task scheduling in mobile edge computing, in: 2020 7th International Conference on Information Science and Control Engineering (ICISCE), IEEE, 2020, pp. 426–431, https://doi.org/10.1109/ICISCE50968.2020.00095
[86] Q. Peng, H. Jiang, M. Chen, J. Liang, Y. Xia, Reliability-aware and deadline-constrained workflow scheduling in mobile edge computing, in: 2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC), 2019, pp. 236–241, https://doi.org/10.1109/ICNSC.2019.8743291
[87] N. Subramanian, A. Jeyaraj, Recent security challenges in cloud computing, Comput. Electr. Eng. 71 (2018) 28–42.
[88] L. Li, C. Zhou, P. Cong, Y. Shen, J. Zhou, T. Wei, Makespan and security-aware workflow scheduling for cloud service cost minimization, IEEE Trans. Cloud Comput. 12 (2) (2024) 609–624.
[89] A. Taghinezhad-Niar, J. Taheri, Security, reliability, cost, and energy-aware scheduling of real-time workflows in compute-continuum environments, IEEE Trans. Cloud Comput. 12 (3) (2024) 954–965.
[90] M. Hosseini Shirvani, A.M. Rahmani, A. Sahafi, An iterative mathematical decision model for cloud migration: a cost and security risk approach, Softw.: Pract. Exper. 48 (3) (2018) 449–485.
[91] Y. Han, Z. Zhao, J. Mo, C. Shu, G. Min, Efficient task offloading with dependency guarantees in ultra-dense edge networks, in: 2019 IEEE Global Communications Conference (GLOBECOM), IEEE, 2019, pp. 1–6, https://doi.org/10.1109/GLOBECOM38437.2019.9013142
[92] M. Zivkovic, T. Bezdan, I. Strumberger, N. Bacanin, K. Venkatachalam, Improved Harris Hawks optimization algorithm for workflow scheduling challenge in cloud–edge environment, in: Computer Networks, Big Data and IoT: Proceedings of ICCBI 2020, Springer, 2021, pp. 87–102.
[93] N. Bacanin, M. Zivkovic, T. Bezdan, K. Venkatachalam, M. Abouhawwash, Modified firefly algorithm for workflow scheduling in cloud-edge environment, Neural Comput. Appl. 34 (11) (2022) 9043–9068.
[94] A.A. Al-Habob, O.A. Dobre, A.G. Armada, S. Muhaidat, Task scheduling for mobile edge computing using genetic algorithm and conflict graphs, IEEE Trans. Veh. Technol. 69 (8) (2020) 8805–8819, https://doi.org/10.1109/TVT.2020.2995146
[95] J. Yuan, H. Xiao, Z. Shen, T. Zhang, J. Jin, Elect: energy-efficient intelligent edge–cloud collaboration for remote IOT services, Futur. Gener. Comput. Syst. 147 (2023) 179–194.
[96] J. Yan, S. Bi, Y.J.A. Zhang, Offloading and resource allocation with general task graph in mobile edge computing: a deep reinforcement learning approach, IEEE Trans. Wireless Commun. 19 (8) (2020) 5404–5419, https://doi.org/10.1109/TWC.2020.2993071
[97] H. Xiao, C. Xu, Y. Ma, S. Yang, L. Zhong, G.-M. Muntean, Edge intelligence: a computational task offloading scheme for dependent IOT application, IEEE Trans. Wireless Commun. 21 (9) (2022) 7222–7237, https://doi.org/10.1109/TWC.2022.3156905
[98] J. Fang, D. Qu, H. Chen, Y. Liu, Dependency-aware dynamic task offloading based on deep reinforcement learning in mobile edge computing, IEEE Trans. Netw. Serv. Manag. (2023) 1, https://doi.org/10.1109/TNSM.2023.3319294
[99] M. Li, N. Mao, X. Zheng, T.R. Gadekallu, Computation offloading in edge computing based on deep reinforcement learning, in: Proceedings of International Conference on Computing and Communication Networks: ICCCN 2021, Springer, 2022, pp. 339–353.
[100] J. Wang, J. Hu, G. Min, W. Zhan, A.Y. Zomaya, N. Georgalas, Dependent task offloading for edge computing based on deep reinforcement learning, IEEE Trans. Comput. 71 (10) (2022) 2449–2461, https://doi.org/10.1109/TC.2021.3131040
[101] L. Zhao, E. Zhang, S. Wan, A. Hawbani, A.Y. Al-Dubai, G. Min, A.Y. Zomaya, MESON: a mobility-aware dependent task offloading scheme for urban vehicular edge computing, IEEE Trans. Mob. Comput. 23 (5) (2023) 4259–4272.
[102] D.A. Van Veldhuizen, G.B. Lamont, et al., Evolutionary computation and convergence to a pareto front, in: Late Breaking Papers at the Genetic Programming 1998 Conference, Citeseer, 1998, pp. 221–228.
[103] K. Peng, M. Zhu, Y. Zhang, L. Liu, J. Zhang, V.C.M. Leung, L. Zheng, An energy-and cost-aware computation offloading method for workflow applications in mobile edge computing, Eur. J. Wirel. Commun. Netw. 2019 (2019) 1–15.
[104] F. Li, W.J. Tan, W. Cai, A wholistic optimization of containerized workflow scheduling and deployment in the cloud–edge environment, Simul. Model. Pract. Theory 118 (2022) 102521.
[105] Y.-Y. Cui, D.-G. Zhang, T. Zhang, J. Zhang, M. Piao, A novel offloading scheduling method for mobile application in mobile edge computing, Wireless Netw. 28 (6) (2022) 2345–2363.
[106] L. Pan, X. Liu, Z. Jia, J. Xu, X. Li, A multi-objective clustering evolutionary algorithm for multi-workflow computation offloading in mobile edge computing, IEEE Trans. Cloud Comput. 11 (2) (2023) 1334–1351, https://doi.org/10.1109/TCC.2021.3132175
[107] X. Xu, H. Cao, Q. Geng, X. Liu, F. Dai, C. Wang, Dynamic resource provisioning for workflow scheduling under uncertainty in edge computing environment, Concurr. Comput. Pract. Exp. 34 (14) (2022) e5674.
[108] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, K. Li, Offloading dependent tasks in multi-access edge computing: a multi-objective reinforcement learning approach, Futur. Gener. Comput. Syst. 128 (2022) 333–348.
[109] T. Tang, C. Li, F. Liu, Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning, Comput. Commun. 209 (2023) 78–90.

[110] K. Zhu, Z. Zhang, S. Zeadally, F. Sun, Learning to optimize workflow scheduling for an edge–cloud computing environment, IEEE Trans. Cloud Comput. 12 (3) (2024) 897–912.
[111] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw.: Pract. Exper. 41 (1) (2011) 23–50.
[112] W. Chen, E. Deelman, WorkflowSim: a toolkit for simulating scientific workflows in distributed environments, in: 2012 IEEE 8th International Conference on E-Science, IEEE, 2012, pp. 1–8.
[113] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya, iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Softw.: Pract. Exper. 47 (9) (2017) 1275–1296.
[114] B. Qin, Q. Lei, X. Wang, DGCQN: a RL and GCN combined method for DAG scheduling in edge computing, J. Supercomput. (2024) 1–28.
[115] C. Shan, R. Gao, Q. Han, T. Liu, Z. Yang, J. Zhang, Y. Xia, KCES: a workflow containerization scheduling scheme under cloud-edge collaboration framework, IEEE Internet Things J. 12 (2) (2024) 2026–2042.
[116] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P.J. Maechling, R. Mayani, W. Chen, R.F. Da Silva, M. Livny, et al., Pegasus, a workflow management system for science automation, Futur. Gener. Comput. Syst. 46 (2015) 17–35.
[117] C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, Heterogeneity and dynamicity of clouds at scale: Google trace analysis, in: Proceedings of the Third ACM Symposium on Cloud Computing, 2012, pp. 1–13.
[118] Alibaba, Alibaba cluster data. https://github.com/alibaba/clusterdata (Accessed: 29 October 2025).
[119] M. Mehrabi, S. Shen, V. Latzko, Y. Wang, F.H.P. Fitzek, Energy-aware cooperative offloading framework for inter-dependent and delay-sensitive tasks, in: GLOBECOM 2020-2020 IEEE Global Communications Conference, 2020, pp. 1–6, https://doi.org/10.1109/GLOBECOM42002.2020.9348078
[120] H. Arabnejad, J.G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2013) 682–694.
[121] R. NoorianTalouki, M. Hosseini Shirvani, H. Motameni, A heuristic-based task scheduling algorithm for scientific workflows in heterogeneous cloud computing platforms, J. King Saud Univ. Comput. Inf. Sci. 34 (8, Part A) (2022) 4902–4913. ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2021.05.011
[122] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (Dec 1999) 406–471. ISSN 0360-0300, https://doi.org/10.1145/344588.344618
[123] S. Karami, S. Azizi, F. Ahmadizar, A bi-objective workflow scheduling in virtualized fog-cloud computing using NSGA-II with semi-greedy initialization, Appl. Soft Comput. 151 (2024) 111142.
[124] S. Abrishami, M. Naghibzadeh, D.H.J. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Futur. Gener. Comput. Syst. 29 (1) (2013) 158–169.
[125] C. Shu, Z. Zhao, Y. Han, G. Min, Dependency-aware and latency-optimal computation offloading for multi-user edge computing networks, in: 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2019, pp. 1–9, https://doi.org/10.1109/SAHCN.2019.8824941
[126] C. Shu, Z. Zhao, Y. Han, G. Min, H. Duan, Multi-user offloading for edge computing networks: a dependency-aware and latency-optimal approach, IEEE Internet Things J. 7 (3) (2020) 1678–1689, https://doi.org/10.1109/JIOT.2019.2943373
[127] N. Khaledian, K. Khamforoosh, S. Azizi, V. Maihami, IKH-EFT: an improved method of workflow scheduling using the krill herd algorithm in the fog-cloud environment, Sustain. Comput. Inform. Syst. 37 (2023) 100834.
[128] Z. Zhang, K. Long, J. Wang, F. Dressler, On swarm intelligence inspired self-organized networking: its bionic mechanisms, designing principles and optimization approaches, IEEE Commun. Surv. & Tutorials 16 (1) (2014) 513–537, https://doi.org/10.1109/SURV.2013.062613.00014
[129] N. Eagle, A. Pentland, Reality mining: sensing complex social systems, Pers. Ubiquitous Comput. 10 (4) (2006) 255–268.
[130] M.-Y. Wu, D.D. Gajski, et al., Hypertool: a programming aid for message-passing systems, IEEE Trans. Parallel Distrib. Syst. 1 (3) (1990) 330–343.
[131] Y.-C. Chung, et al., Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors, in: SC Conference, IEEE Computer Society, 1992, pp. 512–521.
[132] S.J. Kim, A general approach to mapping of parallel computations upon multiprocessor architectures, in: Proc. International Conference on Parallel Processing, vol. 3, IEEE Computer Society, 1988.
[133] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197, https://doi.org/10.1109/4235.996017
[134] Y. Yuan, H. Xu, B. Wang, An improved NSGA-III procedure for evolutionary many-objective optimization, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 661–668.
[135] K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, Deep reinforcement learning: a brief survey, IEEE Signal Process. Mag. 34 (6) (2017) 26–38, https://doi.org/10.1109/MSP.2017.2743240
[136] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, Y. Yang, Optimal edge user allocation in edge computing with variable sized vector bin packing, in: International Conference on Service-Oriented Computing, Springer, 2018, pp. 230–245.
[137] H. Wu, W. Knottenbelt, K. Wolter, Y. Sun, An optimal offloading partitioning algorithm in mobile cloud computing, in: International Conference on Quantitative Evaluation of Systems, Springer, 2016, pp. 311–328.
[138] C.-T. Mo, J.-H. Chen, W. Liao, Graph convolutional network augmented deep reinforcement learning for dependent task offloading in mobile edge computing, in: 2023 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2023, pp. 1–6.
[139] Z. Cao, X. Deng, S. Yue, P. Jiang, J. Ren, J. Gui, Dependent task offloading in edge computing using GNN and deep reinforcement learning, IEEE Internet Things J. 11 (12) (2024) 21632–21646.
[140] B. Huang, L. Wang, X. Liu, Z. Huang, Y. Yin, F. Zhu, S. Wang, S. Deng, Reinforcement learning-based online scheduling of multiple workflows in edge environment, IEEE Trans. Netw. Serv. Manag. 21 (5) (2024) 5691–5706, https://doi.org/10.1109/TNSM.2024.3428496
[141] S. Tuli, S.R. Poojara, S.N. Srirama, G. Casale, N.R. Jennings, COSCO: container orchestration using co-simulation and gradient based optimization for fog computing environments, IEEE Trans. Parallel Distrib. Syst. 33 (1) (2021) 101–116.
[142] X. Tang, F. Liu, B. Wang, D. Xu, J. Jiang, Q. Wu, C.L.P. Chen, Workflow scheduling based on asynchronous advantage actor–critic algorithm in multi-cloud environment, Expert Syst. Appl. 258 (2024) 125245.
[143] H.D. Karatza, G.L. Stavrinides, Resource allocation and aging priority-based scheduling of linear workflow applications with transient failures and selective imprecise computations, Cluster Comput. 27 (4) (2024) 5473–5488.
[144] H. Sun, H. Yu, G. Fan, L. Chen, QoS-aware task placement with fault-tolerance in the edge-cloud, IEEE Access 8 (2020) 77987–78003.
[145] S. Meng, Q. Li, T. Wu, W. Huang, J. Zhang, W. Li, A fault-tolerant dynamic scheduling method on hierarchical mobile edge cloud computing, Comput. Intell. 35 (3) (2019) 577–598.
[146] R. Xie, Q. Tang, S. Qiao, H. Zhu, F.R. Yu, T. Huang, When serverless computing meets edge computing: architecture, challenges, and open issues, IEEE Wirel. Commun. 28 (5) (2021) 126–133.
[147] M. Hosseini Shirvani, Y. Ramzanpoor, Multi-objective qos-aware optimization for deployment of IOT applications on cloud and fog computing infrastructure, Neural Comput. Appl. 35 (26) (2023) 19581–19626.
[148] S. Kim, S. Moon, R. Tabrizi, N. Lee, M.W. Mahoney, K. Keutzer, A. Gholami, An LLM compiler for parallel function calling, in: Forty-First International Conference on Machine Learning, 2024.
[149] X. Tan, Y. Jiang, Y. Yang, H. Xu, Teola: Towards end-to-end optimization of LLM-based applications, arXiv preprint arXiv:2407.00326, 2024.