



GraphMatch: A graph-based trust-aware framework for secure multi-objective task scheduling in heterogeneous edge computing

Wenjuan Li^{a,b,*}, Qifei Zhang^c, Dingyu Yang^b, Chengjie Pan^a, Ben Wang^a, Shuiguang Deng^b

^a School of Information Science and Technology, Hangzhou Normal University, Hangzhou, 311121, China

^b Department of Computer Science and Technology, Zhejiang University, Hangzhou, 310058, China

^c School of Software Engineering, Zhejiang University, Ningbo, 315048, China

ARTICLE INFO

Keywords:

Edge computing
Task scheduling
Graph neural network
Trust management
Multi-objective optimization
Security-aware

ABSTRACT

With the rapid development of IoT and 5G technologies, edge computing has become a critical infrastructure for intelligent applications. However, the openness and heterogeneity of edge computing environments pose severe security challenges: malicious edge nodes may steal sensitive data or tamper with computation results. Existing task scheduling algorithms primarily focus on performance optimization, lacking comprehensive consideration of security and trust. To address these challenges, this paper proposes GraphMatch, a security-aware task scheduling framework based on Graph Convolutional Network (GCN). GraphMatch models task scheduling as a constrained graph matching problem and achieves multi-objective optimization of security, trust, and performance through four synergistic modules: (1) Multi-hop Trust Propagation that expands trust coverage by discovering indirect trust paths; (2) Trust-Guided GCN that learns optimal task-device matching by fusing five scoring mechanisms; (3) NSGA-III-based Multi-objective Optimization that explores Pareto optimal solutions among makespan, trust, and security objectives; (4) Laplacian Load Balancing that achieves workload distribution through spectral graph diffusion. Experimental results demonstrate that GraphMatch achieves a trust score of 0.8869, representing a 10.0% improvement over the state-of-the-art SecDS algorithm. GraphMatch attains 100% malicious device avoidance rate and sensitive task protection rate while maintaining competitive makespan performance. Scalability experiments verify the algorithm's stability across 50–250 device scales, and robustness experiments confirm its reliability under 5%–30% malicious node ratios. GraphMatch provides an effective solution for security-sensitive task scheduling in edge computing.

1. Introduction

Task scheduling in edge computing systems faces increasingly severe security challenges. With the rapid proliferation of edge infrastructure, the number of global IoT devices is projected to reach 75 billion by 2025 [1,2]. The open and distributed nature of edge environments exposes critical security vulnerabilities: malicious edge nodes may steal sensitive data, tamper with computation results, or deliberately delay task execution. Studies show that in open edge settings, the proportion of malicious nodes can reach 10%–40% [3,4], posing a major threat to security-sensitive applications such as medical-data processing, financial-transaction computing and industrial-control systems [5–8].

Despite urgent security needs, existing task-scheduling algorithms still focus primarily on performance optimization, treating security

as a secondary concern. Classical algorithms like HEFT [9] and Min-Min [10] optimize makespan or resource utilization, yet lack trust-evaluation and malicious-node avoidance mechanisms. Recently, trust-aware approaches such as DDTMOA [11], BD-TTS [12], and SecDS [13] have begun to incorporate security considerations, but suffer from three fundamental limitations.

First, trust assessment is overly simplistic. Most current methods rely on trust metrics based solely on direct interaction history, ignoring indirect trust information embedded in the network topology. When direct interactions between a task requester and candidate edge nodes are sparse or absent, which is common in dynamic edge environments, these methods fail to provide reliable trust evaluations, leading to non-optimal or even unsafe scheduling decisions.

Second, optimization objectives are treated in isolation. Existing algorithms usually optimize a single objective (e.g., minimizing latency

* Corresponding author at: School of Information Science and Technology, Hangzhou Normal University, Hangzhou, 311121, China.

E-mail addresses: liwenjuan@hznu.edu.cn, liellie@163.com (W. Li).

¹ Associate Professor, School of Information Science and Technology, Hangzhou Normal University, Hangzhou 311121, China & Visiting Scholar, Department of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China.

<https://doi.org/10.1016/j.future.2026.108580>

Received 26 January 2026; Received in revised form 29 April 2026; Accepted 1 May 2026

Available online 8 May 2026

0167-739X/© 2026 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

or maximizing trust) or adopt weighted-sum approaches that require manual tuning. These methods cannot effectively handle the inherent trade-offs among makespan, trust degree, and security, and perform poorly when these objectives conflict.

Third, representation-learning capability is limited. Current heuristic- and rule-driven approaches struggle to capture the complex structural relationships between task dependencies and device network topology, and are unable to learn optimal task-device matching patterns from the heterogeneous graph structures characteristic of edge computing environments.

These limitations reveal an urgent need for a fundamentally new approach, one that can propagate trust across the network topology to overcome sparse direct interactions, jointly optimize multiple conflicting objectives, and leverage graph neural networks to learn complex task-device matching relationships.

To address these challenges, we propose GraphMatch: a graph-based, trust-aware, secure multi-objective task-scheduling framework for heterogeneous edge computing. GraphMatch formulates task scheduling as a constrained graph-matching problem and achieves joint optimization of security, trust degree, and performance through four synergistic modules: (1) a multi-hop trust-propagation module that iteratively discovers indirect trust paths to extend trust coverage; (2) a Trust-Guided Graph Convolutional Network (GCN) module that embeds the trust matrix into the graph-convolutional message-passing process to learn optimal task-device matches; (3) an NSGA-III multi-objective optimization module that explores the Pareto front among makespan, trust degree, and security objectives; and (4) a Laplacian load-balancing module that uses spectral graph diffusion to smooth workload distribution across devices.

We conducted extensive simulation experiments comparing GraphMatch with six representative algorithms. The main contributions are summarized as follows.

1. We introduce a novel trust-guided graph-matching scheduling paradigm. By modeling edge-computing task scheduling as a graph-matching problem under trust constraints, we design Trust-Guided GCN to achieve a deep fusion of trust information and graph structure, fusing five scoring mechanisms (GCN embedding, performance, trust, type compatibility, and safety) to learn optimal task-device assignments. To the best of our knowledge, this is the first work that combines trust-guided GCN with multi-objective optimization for edge-computing task scheduling. Compared with the state-of-the-art baseline SecDS, GraphMatch improves the average trust score by 10.0%.
2. We devise a multi-hop trust-propagation mechanism to break the sparse-interaction bottleneck. Grounded in the transitivity of trust, an iterative algorithm discovers indirect trust paths and provides a theoretical coverage lower bound (Proposition 1), effectively overcoming insufficient direct interactions in dynamic environments. Experiments show that this mechanism raises average trust coverage by approximately 75%.
3. We construct a multi-objective evolutionary optimization framework with embedded hard security constraints. By integrating hard security constraints and an intelligent security-aware mutation strategy into NSGA-III, we explore the Pareto-optimal solution set while guaranteeing 100% malicious-device avoidance for sensitive tasks and maintaining competitive makespan performance across all tested scenarios.
4. We propose a topology-aware load-balancing method based on spectral graph theory. Leveraging the heat-diffusion property of the Laplacian matrix, it achieves adaptive and smooth workload distribution across the network topology while strictly respecting security and trust constraints, outperforming traditional greedy load-balancing strategies.

The remainder of this paper is organized as follows: Section 2 reviews related work; Section 3 formalizes the problem; Section 4 details the design and implementation of GraphMatch; Section 5 presents experimental results and analysis; Section 6 concludes the paper and outlines future research directions.

2. Related works

This section reviews research progress from three perspectives: edge-computing task scheduling, trust management and security-aware scheduling, and GNN-based scheduling methods. We analyze the core challenges faced by existing approaches and clarify the research positioning of this paper.

2.1. Edge-computing task scheduling

Task scheduling is a cornerstone problem in edge computing, aiming to assign computational tasks to edge devices in a manner that optimizes system performance.

Classical heuristic baselines. Although HEFT [9] and Min-Min [10] were originally proposed for heterogeneous distributed computing, they remain widely adopted as strong baselines in edge-computing scheduling research due to their theoretical rigor and ease of implementation. HEFT adopts upward-rank priority ordering and the earliest-finish-time strategy to schedule DAG-based tasks, while Braun et al. [10] systematically compared eleven static heuristics including Min-Min and Max-Min, establishing a theoretical foundation for independent-task scheduling. However, these methods assume a fully trustworthy execution environment and do not consider the security challenges specific to open edge-computing scenarios.

Deep reinforcement-learning approaches. Recently, deep reinforcement learning (DRL) has demonstrated powerful sequential decision-making capability in MEC task scheduling. Chen et al. [14] proposed DRL-MEC, which uses a DDPG architecture to jointly optimize latency and energy consumption in computation offloading. Mao et al. [15] introduced Decima, which encodes task-DAG structures with graph neural networks and significantly outperforms hand-crafted heuristics in data-processing cluster scheduling. Tao et al. [16] investigated multi-agent cooperation for computing power scheduling in UAV-empowered aerial edge-computing systems, formulating the joint optimization of energy efficiency and service fairness via multi-agent DRL, demonstrating the potential of cooperative DRL in heterogeneous edge environments. Khan et al. [17] proposed EMDTORA, which leverages graph-enabled DRL for energy-aware dependent task offloading and resource allocation in multi-user MEC, modeling inter-task dependencies as graphs to improve scheduling quality.

Nevertheless, all aforementioned methods assume a trustworthy execution environment. Studies show that the proportion of malicious nodes in open edge settings can reach 10%–40% [3,4]; malicious devices may steal sensitive data, tamper with computation results, or launch denial-of-service attacks [5,6]. A survey by Xiao et al. [5] points out that edge computing faces diverse security threats, including DDoS attacks, side-channel attacks, and malware injection, that render purely performance-oriented scheduling strategies incapable of guaranteeing service quality and data security in security-sensitive scenarios.

2.2. Trust management and security-aware scheduling

To address the security challenges of edge computing, researchers have pursued two complementary directions: trust-management architectures and trust-aware scheduling mechanisms.

Distributed trust models. Trust management has a long-standing foundation in P2P networks and distributed systems. Jøsang et al. [18] systematically surveyed trust and reputation systems for online services, elaborating computational models and propagation mechanisms

for trust. Kamvar et al. [19] introduced EigenTrust, which iteratively computes global trust values via eigenvector decomposition and became a milestone in trust-propagation research. In IoT scenarios, Bao and Chen [20] proposed a dynamic trust-management protocol that synthesizes direct observations and indirect recommendations for trust assessment. While these works lay the theoretical groundwork for trust management in edge computing, they are not deeply integrated with task-scheduling decisions.

Trust-aware task scheduling. Recently, researchers have begun to incorporate trust evaluation into scheduling decisions. Tong et al. [11] presented DDTMOA, which constructs a composite trust metric by weighting identity trust, behavior trust, and capability trust, and then uses DDQN for scheduling. Wang et al. [12] proposed BD-TTS, leveraging blockchain to store trust records immutably; however, the consensus mechanism incurs latency and storage overhead that limits scalability. Long et al. [13] introduced SecDS, which classifies task-security risk into low, medium, and high levels and employs a DAG-topology-based scheduling strategy.

Despite incorporating trust and security considerations, existing methods still face three core challenges:

1. **Trust sparsity problem.** Most current approaches compute trust solely from direct interaction history. In dynamic edge environments, however, direct device-to-device contacts are often scarce or non-existent. For instance, newly joined devices lack historical records, and geographically dispersed devices rarely meet. When direct-trust coverage is insufficient, the scheduler lacks adequate information for reliable decisions and falls back to random selection or overly conservative policies. Although EigenTrust [19] and similar works study trust propagation, they target P2P file-sharing rather than edge-computing scheduling.
2. **Multi-objective conflicts.** Edge scheduling must simultaneously optimize makespan, trust degree, security, etc. These objectives inherently conflict: choosing high-trust devices may increase latency, while avoiding every risk can drive resource utilization down. Existing schemes typically merge objectives through weighted sums (e.g., the fixed 0.3:0.4:0.3 weights in DDTMOA [11]), forcing decision makers to specify weights a priori and failing to explore the full trade-off surface, so users with different preferences receive few alternative solutions.
3. **Rigid safety demands.** For security-sensitive applications such as medical-data processing or financial-transaction computing, sensitive tasks must never be placed on malicious devices; this is an inflexible hard constraint, not an objective to be traded off. Yet most current treatments cast security as a soft constraint or penalty term (e.g., the risk-level model of SecDS [13]), unable to provide absolute safety guarantees.

2.3. GNN-based scheduling

Task scheduling in edge computing is inherently a graph-structured problem: devices form a network topology, tasks exhibit DAG dependencies, and task-device assignments can be viewed as bipartite graph matching. Graph Neural Networks (GNNs) are a natural fit for such structured data, as they learn node representations by aggregating neighborhood information through message passing. Representative architectures include GCN [21], which performs spectral-domain convolutions, and GraphSAGE [22], which inductively samples and aggregates features to scale to large graphs.

GNN-based task scheduling and offloading. Based on these architectures, recent works have applied GNNs to various scheduling and combinatorial optimization problems. Mao et al. [15] introduced Decima, which encodes task-DAG structures via GNNs and learns scheduling policies that significantly outperform hand-crafted heuristics. Park et al. [23] model the job-shop scheduling problem as a disjunctive graph, employ a GNN for state representation, and optimize the policy

with PPO. More recently, Liao et al. [24] proposed a GCN-enabled task offloading scheme for industrial image recognition in digital twin edge networks, where GCN extracts DAG subtask dependencies to generate optimal execution priorities, combined with multi-agent DRL for dynamic offloading decisions. Khan et al. [17] further demonstrated graph-enabled DRL for dependent task offloading with energy-aware resource allocation in multi-user MEC. In the domain of trust management, Awan et al. [25] introduced TrustAware-GNN, which integrates multi-dimensional trust (direct, indirect, temporal, and contextual) into a GNN framework for IoT anomaly detection, with trust values modulating edge weights for trust-adaptive message propagation. While these works demonstrate the power of GNNs for graph-structured scheduling and trust evaluation respectively, none of them combines trust-guided representation learning with security-constrained task scheduling.

Multi-objective evolutionary optimization. When the number of conflicting objectives exceeds two, evolutionary algorithms are the dominant approach. Deb et al. [26] proposed NSGA-II, which maintains diversity via non-dominated sorting and crowding-distance selection. Deb and Jain [27] advanced to NSGA-III, using reference points to handle problems with three or more objectives, making it well-suited for the joint optimization of makespan, trust, and security in edge scheduling. Zhang and Li [28] introduced MOEA/D, decomposing the multi-objective problem into cooperative single-objective sub-problems.

In summary, existing GNN-based scheduling methods exhibit three critical gaps: (1) *no trust propagation*—none of the above GNN scheduling works incorporates trust information into the graph-convolutional message-passing process; (2) *no hard security constraints*—current approaches treat security as a soft penalty or ignore it entirely, failing to guarantee that sensitive tasks are never assigned to malicious devices; and (3) *no Pareto-based multi-objective optimization*—most GNN scheduling methods optimize a single objective or use fixed weighted sums, unable to explore the full trade-off surface among makespan, trust, and security. GraphMatch is designed to address all three gaps simultaneously.

2.4. Research positioning and contributions

Table 1 compares representative existing methods from the perspective of practical requirements. As shown, none of them can simultaneously and effectively address the three core challenges: trust sparsity, multi-objective conflicts, and rigid safety guarantees.

To address these three challenges, we propose GraphMatch, which is equipped with the following modules: (1) a multi-hop trust-propagation module that iteratively discovers indirect trust paths, giving the scheduler reliable trust estimates even when direct interactions are scarce; (2) a Trust-Guided GCN module that fuses trust information with the graph structure to learn optimal task-device matches; (3) an NSGA-III evolutionary engine that explores the Pareto front, offering decision makers a diverse set of trade-off solutions without preset weights; (4) hard security constraints that treat malicious-device avoidance as an inviolable requirement, ensuring sensitive tasks are never placed on malicious nodes.

3. Problem modeling and formal description

In this section, we provide a formal model of the security-aware task-scheduling problem in edge-computing environments. We first introduce the key notation, then present the system model, optimization objectives, and constraints, and finally analyze the computational complexity of the problem.

3.1. Notation

For clarity, Table 2 summarizes the main symbols used throughout the paper.

To avoid symbol clashes, we use R for the trust (reputation) matrix and T for the task set.

Table 1
Comparison of existing methods on the three core challenges.

Method	Handles trust sparsity	Handles multi-objective conflict	Safety guarantee
HEFT [9]	× no trust	× single objective	× no security
DRL-MEC [14]	× no trust	weighted sum	× no security
EigenTrust [19]	✓trust propagation	N/A (not for scheduling)	N/A
DDTMOA [11]	× direct trust only	weighted sum (fixed weights)	× soft constraint
BD-TTS [12]	× direct trust only	weighted sum	partial (blockchain verify)
SecDS [13]	× direct trust only	weighted sum	partial (risk levels)
Decima [15]	× no trust	× single objective	× no security
GraphMatch (Ours)	✓multi-hop trust propagation	✓Pareto optimization	✓hard constraint guarantee

Table 2
Key notation.

Symbol	Meaning
$D = \{d_1, \dots, d_n\}$	Set of edge devices (n in total)
$T = \{t_1, \dots, t_m\}$	Set of tasks (m in total)
G_D, G_T	Device topology graph and task DAG
$R, R^{(mh)} \in [0, 1]^{n \times n}$	Direct and multi-hop trust matrices
$\pi : T \rightarrow D$	Task-to-device assignment

3.2. System model

Edge device model. Consider an edge computing system composed of n heterogeneous edge devices, denoted by the set $D = \{d_1, d_2, \dots, d_n\}$. Each device d_i is characterized by a 4-tuple:

$$d_i = \langle c_i, m_i, b_i, mal_i \rangle \quad (1)$$

where c_i is the computational capacity (GFLOPS), m_i is the memory size (GB), b_i is the maximum network interface capacity of device d_i (Mbps), representing the physical upper bound of its communication hardware (see B_{ij} below for the effective link bandwidth), and $mal_i \in \{0, 1\}$ is a malicious flag (1 indicates malicious). The set of malicious devices is defined as $M = \{d_i \in D : mal_i = 1\}$.

The device network topology is represented by an undirected graph $G_D = (D, E_D)$, where $E_D \subseteq D \times D$ is the set of communication links. For a link $(d_i, d_j) \in E_D$, its effective communication bandwidth is denoted by B_{ij} , which satisfies $B_{ij} \leq \min(b_i, b_j)$ due to the interface capacity constraint of both endpoints. In our simulator, B_{ij} is assigned during topology generation to reflect realistic heterogeneous link conditions. The communication latency is defined as

$$Comm(d_i, d_j, s) = \frac{s}{B_{ij}} \quad (2)$$

where s is the amount of data to be transmitted.

Task model. The system needs to schedule m compute tasks, denoted by the set $T = \{t_1, t_2, \dots, t_m\}$. Each task t_j is described by a 5-tuple:

$$t_j = \langle w_j, r_j, s_j, src_j, sen_j \rangle \quad (3)$$

where w_j is the computational demand (in millions of CPU cycles), r_j is the memory requirement (GB), s_j is the input-data size (MB), $src_j \in D$ is the device that submitted the task, and $sen_j \in \{0, 1\}$ is a sensitivity flag (1 indicates sensitive data).

Task dependencies are captured by a directed acyclic graph $G_T = (T, E_T)$, where $E_T \subseteq T \times T$ is the set of dependency edges. If $(t_i, t_j) \in E_T$, task t_j depends on the output of task t_i , i.e., t_j can start only after t_i has finished. The predecessor set of task t_j is defined as

$$pred(t_j) = \{t_i \in T : (t_i, t_j) \in E_T\} \quad (4)$$

Execution time model. The execution time of task t_j on device d_i is defined as

$$Exec(t_j, d_i) = \frac{w_j}{c_i} \quad (5)$$

Taking task dependencies and data transfer into account, when task t_j is assigned to device d_i , its start and finish times are

$$ST(t_j) = \max_{t_k \in pred(t_j)} (FT(t_k) + Comm(\pi(t_k), d_i, s_{kj})) \quad (6)$$

$$FT(t_j) = ST(t_j) + Exec(t_j, d_i) \quad (7)$$

where s_{kj} is the amount of data that task t_k must send to t_j , and $\pi(t_k)$ denotes the device to which t_k is assigned. For entry tasks with no predecessors, $ST(t_j) = 0$.

Trust model. Inter-device trust relationships are represented by a trust matrix $R \in [0, 1]^{n \times n}$, where $R[i, j]$ indicates the degree of trust that device d_i places in device d_j . The trust matrix has the following properties:

- **Property 1 (Self-trust):** $\forall d_i \in D : R_{ii} = 1$
- **Property 2 (Asymmetry):** In general, $R_{ij} \neq R_{ji}$
- **Property 3 (Non-negativity):** $\forall i, j : 0 \leq R_{ij} \leq 1$

The initial trust matrix R contains only trust values derived from direct interactions; for device pairs without direct interaction, $R[i, j]$ is initialized to a default value ρ_0 (set to 0.5 in this paper). Section 4 will present a multi-hop trust-propagation algorithm that expands trust coverage through indirect paths.

3.3. Optimization objectives

The goal of task scheduling is to find an assignment function $\pi : T \rightarrow D$ that maps each task to a device. This paper considers the following three optimization objectives.

3.3.1. Minimize makespan

Makespan is defined as the latest finish time among all tasks and reflects the overall execution efficiency of the schedule:

$$f_1(\pi) = \max_{t_j \in T} FT(t_j) \quad (8)$$

The objective is to minimize $f_1(\pi)$.

3.3.2. Maximize average trust degree

The average trust degree is defined as the mean trust value of all task assignments, indicating the overall trustworthiness of the schedule:

$$f_2(\pi) = \frac{1}{m} \sum_{t_j \in T} R_{src_j, \pi(t_j)} \quad (9)$$

where $R_{src_j, \pi(t_j)}$ denotes the trust that the source device of task t_j places in the device to which the task is assigned. The objective is to maximize $f_2(\pi)$.

3.3.3. Maximize security

Security is defined as the proportion of tasks that are not assigned to malicious devices:

$$f_3(\pi) = 1 - \frac{|\{t_j \in T : \pi(t_j) \in M\}|}{m} \quad (10)$$

The objective is to maximize $f_3(\pi)$.

In summary, the multi-objective optimization problem studied in this paper can be formally stated as

$$\min_{\pi} (f_1(\pi), -f_2(\pi), -f_3(\pi)) \quad (11)$$

subject to the following constraints that any feasible schedule must satisfy:

Constraint 1 (Resource): The memory demand of a task must not exceed the available memory of the assigned device:

$$\forall t_j \in T : r_j \leq m_{\pi(t_j)} \quad (12)$$

Constraint 2 (Dependency): A task can start only after all its predecessors have finished:

$$\forall (t_i, t_j) \in E_T : ST(t_j) \geq FT(t_i) + Comm(\pi(t_i), \pi(t_j), s_{ij}) \quad (13)$$

Constraint 3 (Security): Sensitive tasks are forbidden from being assigned to malicious devices:

$$\forall t_j \in T : sen_j = 1 \rightarrow \pi(t_j) \notin M \quad (14)$$

Constraint 4 (Trust-threshold): A task can be assigned only to a device whose trust value exceeds threshold τ :

$$\forall t_j \in T : R_{src_j, \pi(t_j)} \geq \tau \quad (15)$$

Constraint 3 is a hard constraint and must be strictly satisfied; the threshold τ in Constraint 4 is adjustable and is set to 0.3 by default in this paper. The proposed problem generalizes the classical Multiprocessor Scheduling Problem and is therefore NP-hard [29]; we accordingly design GraphMatch as a heuristic to obtain near-optimal solutions efficiently.

4. System architecture and algorithm design

This section presents the overall system architecture and the design of the core modules of the GraphMatch. GraphMatch adopts a four-stage pipelined architecture that successively realizes security-aware task scheduling through (1) multi-hop trust propagation, (2) trust-guided graph convolutional networks, (3) multi-objective evolutionary optimization, and (4) load balancing.

4.1. System architecture

GraphMatch aims to solve the multi-objective optimization problem defined in Section 3, including: (1) Security guarantee: ensure that sensitive tasks are never assigned to malicious devices (hard constraint), while also minimizing the assignment of ordinary tasks to malicious devices (soft constraint); (2) Trust-coverage extension: discover indirect trust paths through multi-hop trust propagation, overcoming insufficient trust evaluation caused by sparse direct interactions; (3) Intelligent matching learning: exploit graph convolutional networks to learn structured representations of tasks and devices, capturing complex matching patterns; (4) Multi-objective trade-off: adopt the NSGA-III algorithm to explore the Pareto front, providing decision-makers with a diverse set of compromise solutions; (5) Load balancing: achieve smooth workload distribution across devices via spectral graph diffusion, improving overall system throughput.

GraphMatch adopts a four-stage pipelined architecture, as shown in Fig. 1.

Stage 1 (Trust Propagation): Takes the direct trust matrix R as input and outputs the multi-hop trust matrix $R^{(mh)}$, expanding trust coverage.

Stage 2 (Representation Learning): Takes the device graph, task graph, and $R^{(mh)}$ as input. Through Trust-Guided GCN, it learns embedding representations and outputs the task-device matching score matrix S .

Stage 3 (Multi-objective Optimization): Generates initial solutions based on matching scores S , explores the Pareto front through NSGA-III evolutionary optimization, and outputs a set of candidate solutions.

Stage 4 (Load Balancing): Performs load balancing adjustments on candidate solutions and outputs the final assignment scheme A^* .

4.2. Core algorithms design

4.2.1. Multi-hop trust-propagation module

In edge-computing environments, direct trust relationships between devices are usually sparse. When the source device of a task has no direct trust link to a candidate target device, traditional methods fall back to a low default value, yielding unreliable scheduling decisions. The multi-hop trust-propagation module enlarges the trust-coverage area by discovering indirect trust paths.

Let the direct-trust coverage rate be

$$Coverage(R) = \frac{|\{(i, j) : R_{ij} > \rho_0, i \neq j\}|}{n(n-1)} \quad (16)$$

where ρ_0 is the default trust value. In practice $Coverage(R)$ is often below 30%, leaving a large fraction of task-device pairs without dependable trust estimates.

Multi-hop trust propagation rests on the transitivity of trust: if device d_i trusts d_k and d_k trusts d_j , then d_i can derive an indirect trust in d_j . Iterative propagation substantially enlarges the trust-coverage area. The update rule for h -hop trust propagation is defined as

$$R_{ij}^{(h)} = \max \left(R_{ij}^{(h-1)}, \max_{k \neq i, j} \min \left(R_{ik}^{(h-1)}, R_{kj}^{(h-1)} \right) \cdot \gamma^h \right) \quad (17)$$

where $R^{(0)} = R$ is the initial direct-trust matrix, $\gamma \in (0, 1)$ is a decay factor that captures the drop of trust with hop count, $\min(\cdot, \cdot)$ adopts a conservative strategy by taking the minimum trust value along a path, and $\max$ preserves any previously obtained higher trust value.

After h hops, the algorithm outputs the multi-hop trust matrix $R^{(mh)} = R^{(H)}$. Pseudocode for the multi-hop trust-propagation procedure is given in Algorithm 1.

Proposition 1 (Trust Coverage Monotonicity and Growth). *The multi-hop trust coverage rate is monotonically non-decreasing in H , i.e., $Coverage(R^{(h)}) \leq Coverage(R^{(h+1)})$ for all $h \geq 0$. Furthermore, for a connected graph G_D , the coverage rate satisfies*

$$Coverage(R^{(H)}) \geq 1 - (1 - Coverage(R^{(0)}))^H \quad (18)$$

Proof. Monotonicity follows directly from the update rule in Eq. (17): $R_{ij}^{(h)}$ is updated via $\max(\cdot, \cdot)$, so trust values never decrease across iterations, and any previously covered pair remains covered.

For the lower bound, let $U_h = 1 - Coverage(R^{(h)})$ denote the uncovered fraction at hop h . In a connected graph, for any uncovered pair (d_i, d_j) at hop $h-1$, there exists an intermediate device d_k providing an indirect path. By Eq. (17), each hop reduces the uncovered fraction by at least a factor of $U_0 = 1 - Coverage(R^{(0)})$, yielding $U_H \leq U_0^H$ and thus the stated bound. $\square$

Time complexity of Algorithm 1 is $O(n^3 \cdot H)$, where n is the number of devices and H is the maximum hop count. Space complexity is $O(n^2)$. By default we set $H = 3$, $\gamma = 0.8$, and $\theta = 0.9$. A larger H introduces noise, while a smaller γ causes indirect trust to decay too rapidly.

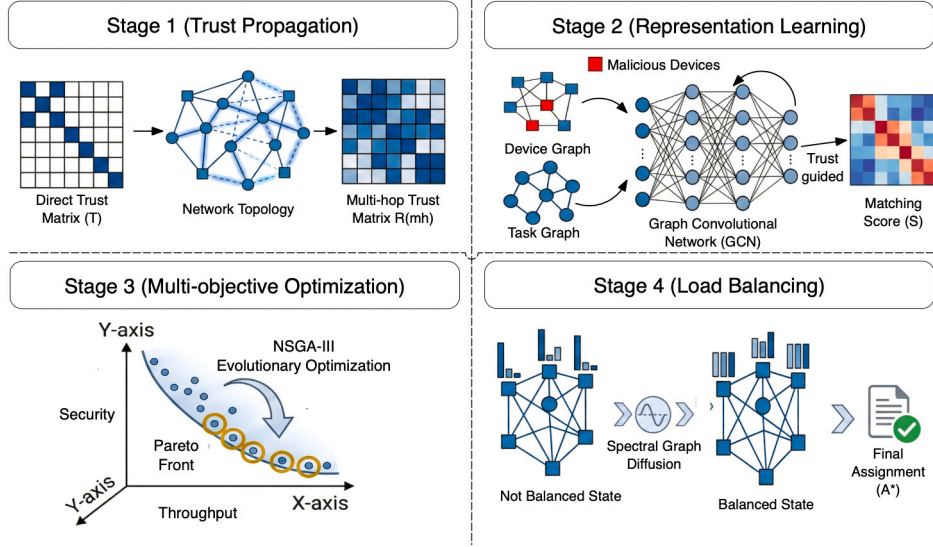


Fig. 1. The overall architecture of GraphMatch.

Algorithm 1 Multi-hop Trust Propagation

Require: Direct trust matrix R , max hops H , decay factor γ , threshold θ

Ensure: Multi-hop trust matrix $R^{(mh)}$

```

1:  $R^{(mh)} \leftarrow R$ 
2:  $n \leftarrow |V_D|$ 
3: for  $h = 1$  to  $H$  do
4:   for  $i = 1$  to  $n$  do
5:     for  $j = 1$  to  $n$  do
6:       if  $i \neq j$  and  $R^{(mh)}[i, j] < \theta$  then
7:          $max\_indirect \leftarrow 0$ 
8:         for  $k = 1$  to  $n$  do
9:           if  $k \neq i$  and  $k \neq j$  then
10:             $path\_trust \leftarrow \min(R^{(mh)}[i, k], R^{(mh)}[k, j]) \times \gamma^h$  {cf. Eq. (17)}
11:             $max\_indirect \leftarrow \max(max\_indirect, path\_trust)$ 
12:          end if
13:        end for
14:         $R^{(mh)}[i, j] \leftarrow \max(R^{(mh)}[i, j], max\_indirect)$ 
15:      end if
16:    end for
17:  end for
18: end for
19: return  $R^{(mh)}$ 

```

4.2.2. Trust-Guided GCN module

Conventional task-scheduling algorithms usually rely on simple heuristic rules (e.g., the priority list in HEFT) and suffer from two key limitations: (i) they fail to exploit the topological structure of the device network, and (ii) they cannot effectively integrate trust relationships into joint decision-making. To address these shortcomings, we propose the Trust-Guided GCN module. Its core innovation is to integrate the trust matrix deeply into the message-passing process of Trust-Guided GCN, so that information from high-trust neighbors is amplified while information from low-trust neighbors is suppressed. In this way, the module learns high-dimensional embeddings that simultaneously encode capability features and trust relations.

Device-embedding network. The device network graph G_D has adjacency matrix $A_D \in \{0, 1\}^{n \times n}$ and initial device feature matrix

$X_D \in \mathbb{R}^{n \times f_d}$, where f_d is the feature dimension containing normalized attributes such as compute capacity, memory size, and network bandwidth.

Standard GCN message passing treats all neighbors equally, balancing their contributions only through degree normalization. In a secure edge computing environment, however, we want high-trust neighbors to carry larger propagation weights while low-trust neighbors are down-weighted. The core innovation of Trust-Guided GCN is therefore to inject the multi-hop trust matrix $R^{(mh)}$ into the message-passing process. Concretely, we perform an element-wise (Hadamard) product between the adjacency matrix and the trust matrix to form a trust-weighted adjacency matrix:

$$A_D^{(trust)} = \tilde{A}_D \odot R^{(mh)} \quad (19)$$

where $\odot$ denotes element-wise multiplication. Through this operation, the low-trust neighbors thus have their adjacency weights attenuated, whereas weights of high-trust neighbors remain unchanged.

With this trust-weighted adjacency matrix, the propagation rule of the device embedding network is defined as:

$$H_D^{(l+1)} = \sigma \left(\tilde{D}_D^{-1/2} (A_D \odot R^{(mh)}) \tilde{D}_D^{-1/2} H_D^{(l)} W_D^{(l)} \right) \quad (20)$$

where $\tilde{D}_D$ is the degree matrix of $A_D^{(trust)}$, σ is an activation function, and $W_D^{(l)}$ are learnable parameters. Starting from $H_D^{(0)} = X_D$, after L layers we obtain the device embedding matrix $H_D = H_D^{(L)} \in \mathbb{R}^{n \times d}$, with d being the embedding dimension.

Task-embedding network. Let the adjacency matrix of the task DAG G_T be $A_T \in \{0, 1\}^{m \times m}$ and the initial task-feature matrix be $X_T \in \mathbb{R}^{m \times f_t}$, which contains attributes such as computational demand, memory demand, and the sensitive-data flag. Since trust relationships exist only between devices, not between tasks, the task-embedding network adopts the standard GCN propagation rule:

$$H_T^{(l+1)} = \sigma \left(\tilde{D}_T^{-1/2} \tilde{A}_T \tilde{D}_T^{-1/2} H_T^{(l)} W_T^{(l)} \right) \quad (21)$$

where $\tilde{A}_T$ is the normalized adjacency matrix of the task DAG and $\tilde{D}_T$ is its degree matrix. Starting from $H_T^{(0)} = X_T$, after L layers we obtain the task-embedding matrix $H_T = H_T^{(L)} \in \mathbb{R}^{m \times d}$.

Multi-dimensional matching-score computation. Using the task embeddings H_T and the device embeddings H_D , we compute a comprehensive matching score between task t_j and device d_i . This module combines five dimension scores, their definitions and roles are summarized in Table 3.

The formulas for each score are as follows:

Table 3
Definitions and roles of the five matching scores.

Score type	Symbol	Role
GCN Embedding Score	$s^{(emb)}$	Captures learned complex matching patterns
Performance Score	$s^{(perf)}$	Prefers devices with high computational capacity
Trust Score	$s^{(trust)}$	Prefers highly trusted devices
Type-Compatibility Score	$s^{(type)}$	Matches task type to device type
Safety Score	$s^{(safe)}$	Penalizes malicious devices

GCN Embedding Score: measures the matching degree between task and device in the embedding space via cosine similarity:

$$s_{ji}^{(emb)} = \cos(h_{t_j}, h_{d_i}) = \frac{h_{t_j}^T h_{d_i}}{\|h_{t_j}\| \|h_{d_i}\|} \quad (22)$$

Performance Score: normalized compute capacity, favoring high-performance devices:

$$s_{ji}^{(perf)} = \frac{c_i}{\max_k c_k} \quad (23)$$

Trust Score: multi-hop trust value from the source device of the task to the candidate device:

$$s_{ji}^{(trust)} = R_{src_j, i}^{(mh)} \quad (24)$$

Type-Compatibility Score: predefined compatibility between task type and device type:

$$s_{ji}^{(type)} = \text{Compat}(\text{type}(t_j), \text{type}(d_i)) \quad (25)$$

where $\text{Compat}(\cdot, \cdot) \in [0, 1]$ is a predefined type-compatibility function.

Safety Score: penalizes malicious devices and imposes a hard constraint on sensitive tasks:

$$s_{ji}^{(safe)} = \begin{cases} 1 & \text{if } d_i \notin M \\ 1 - \alpha & \text{if } d_i \in M \text{ and } \text{sen}_j = 0 \\ -\infty & \text{if } d_i \in M \text{ and } \text{sen}_j = 1 \end{cases} \quad (26)$$

where M is the malicious-device set and $\alpha \in (0, 1)$ is a penalty coefficient (default 0.5). Assigning a sensitive task to a malicious device yields $-\infty$, ensuring the assignment is never selected.

The comprehensive matching score is a weighted sum of the five scores:

$$S_{ji} = w_1 s_{ji}^{(emb)} + w_2 s_{ji}^{(perf)} + w_3 s_{ji}^{(trust)} + w_4 s_{ji}^{(type)} + w_5 s_{ji}^{(safe)} \quad (27)$$

with weights $\{w_1, \dots, w_5\}$ set to 0.2 by default.

Algorithm 2 details the implementation of Trust-Guided GCN.

The time complexity of Algorithm 2 is $O(L \cdot (n^2 d + m^2 d) + mnd)$, where L is the number of GCN layers, n is the number of devices, m is the number of tasks, and d is the embedding dimension. The space complexity is $O(nd + md + mn)$.

4.2.3. Security-aware greedy matcher

The greedy matcher produces an initial task-to-device assignment based on the composite scores S output by the GCN module. Its guiding principle is to favor devices with high trust and high performance while never violating security requirements.

The greedy matcher processes tasks in priority order (sensitive tasks first, then by decreasing computational demand). For each task, it filters candidate devices according to three rules: (1) Hard constraint: sensitive tasks are prohibited from malicious devices (enforced by the $-\infty$ safety score in Eq. (26)); (2) Trust threshold: only devices with $R_{src_j, i}^{(mh)} \geq \tau$ are eligible; (3) Load constraint: devices whose projected load exceeds limit L are excluded.

Among eligible candidates, the device maximizing the composite score $S_{ji} - \lambda \cdot (\text{Load}[d]/L)$ is selected, where the second term penalizes overloaded devices. If no candidate satisfies all constraints, the algorithm falls back to the highest-trust non-malicious device. Algorithm 3 details the full procedure.

Algorithm 2 Trust-Guided GCN Matching

Require: Device graph G_D , task graph G_T , multi-hop trust matrix $R^{(mh)}$, GCN layers L , hidden dimension d

Ensure: Matching score matrix $S \in \mathbb{R}^{m \times n}$

```

1:  $H_D^{(0)} \leftarrow X_D$  {Initialize device features}
2:  $H_T^{(0)} \leftarrow X_T$  {Initialize task features}
3:  $A_D^{(trust)} \leftarrow \tilde{A}_D \odot R^{(mh)}$  {Trust-weighted adjacency, cf. Eq. (19)}
4: for  $l = 0$  to  $L - 1$  do
5:    $H_D^{(l+1)} \leftarrow \sigma(\tilde{D}_D^{-1/2} A_D^{(trust)} \tilde{D}_D^{-1/2} H_D^{(l)} W_D^{(l)})$  {cf. Eq. (20)}
6:    $H_T^{(l+1)} \leftarrow \sigma(\tilde{D}_T^{-1/2} \tilde{A}_T \tilde{D}_T^{-1/2} H_T^{(l)} W_T^{(l)})$  {cf. Eq. (21)}
7: end for
8:  $H_D \leftarrow H_D^{(L)}$ ,  $H_T \leftarrow H_T^{(L)}$ 
9: for  $j = 1$  to  $m$  do
10:  for  $i = 1$  to  $n$  do
11:     $s_{emb} \leftarrow \cos(h_{t_j}, h_{d_i})$  {cf. Eq. (22)}
12:     $s_{perf} \leftarrow c_i / \max_k(c_k)$  {cf. Eq. (23)}
13:     $s_{trust} \leftarrow R^{(mh)}[src_j, i]$  {cf. Eq. (24)}
14:     $s_{type} \leftarrow \text{Compat}(\text{type}(t_j), \text{type}(d_i))$  {cf. Eq. (25)}
15:     $s_{safe} \leftarrow \text{SafetyScore}(t_j, d_i, M)$  {cf. Eq. (26)}
16:     $S[j, i] \leftarrow \sum_{k=1}^5 w_k \cdot s_k$  {cf. Eq. (27)}
17:  end for
18: end for
19: return  $S$ 

```

4.2.4. NSGA-III multi-objective optimization module

Task scheduling is a typical multi-objective problem that must simultaneously optimize trust, latency, energy, and other conflicting goals. NSGA-III is a state-of-the-art evolutionary algorithm that can locate the Pareto-optimal set.

Solution Representation and Initialization. Each individual is an assignment vector π of length m , where $\pi[j] \in \{1, \dots, n\}$ denotes the device to which task t_j is assigned. The initial population is generated by the security-aware greedy matcher in Algorithm 3. Tasks are ordered by priority (sensitive first, then largest workload) and each task is placed on the highest-scoring device that satisfies all constraints.

Fitness Evaluation. For every individual π we compute the three objective values defined in Section 3, including $f_1(\pi)$, $f_2(\pi)$, and $f_3(\pi)$.

Security-Aware Mutation. The key novelty of this work within NSGA-III is a mutation operator that preferentially targets tasks assigned to malicious or low-trust devices. The mutation probability for task t_j is

$$P_{mutate}(t_j) = \begin{cases} 1.0 & \text{if } \pi(t_j) \in M \text{ and } \text{sen}_j = 1 \\ 0.8 & \text{if } \pi(t_j) \in M \text{ and } \text{sen}_j = 0 \\ 0.5 & \text{if } R_{src_j, \pi(t_j)}^{(mh)} < \tau \\ p_m & \text{otherwise} \end{cases} \quad (28)$$

where p_m is the base mutation rate and τ is the trust threshold. This guarantees that any sensitive task accidentally assigned to a malicious device is always mutated and corrected.

Algorithm 4 shows the Multi-Objective Optimization Module based on NSGA-III with security-aware mutation.

Algorithm 3 Security-Aware Greedy Matching

Require: Task set $Tasks$, device set $Devices$, matching scores S , trust matrix $R^{(mh)}$, malicious device set M , makespan limit L

Ensure: Task assignment A

```

1:  $A \leftarrow \emptyset$ ;  $Load[d] \leftarrow 0, \forall d \in Devices$ 
2:  $SortedTasks \leftarrow Sort(Tasks, key = (-sensitive, -cpu\_cycles))$ 
3: for each task  $t \in SortedTasks$  do
4:    $Candidates \leftarrow \emptyset$ 
5:   for each device  $d \in Devices$  do
6:     if  $t.sensitive$  and  $d \in M$  then
7:       continue {Hard security constraint, cf. Eq. (26) & Constraint Eq. (14)}
8:     end if
9:     if  $R^{(mh)}[t.source, d] < \tau$  then
10:      continue {Trust threshold, cf. Constraint Eq. (15)}
11:     end if
12:     if  $Load[d] + ExecTime(t, d) > L$  then
13:       continue
14:     end if
15:      $score \leftarrow S[t, d] - \lambda \times (Load[d]/L)$  {cf. Eq. (27)}
16:     if  $d \in M$  then
17:        $score \leftarrow score - penalty$ 
18:     end if
19:      $Candidates \leftarrow Candidates \cup \{(d, score)\}$ 
20:   end for
21:   if  $Candidates \neq \emptyset$  then
22:      $d^* \leftarrow \arg \max \{score \mid (d, score) \in Candidates\}$ 
23:   else
24:      $d^* \leftarrow \arg \max \{R^{(mh)}[t.source, d] \mid d \notin M\}$ 
25:   end if
26:    $A[t] \leftarrow d^*$ 
27:    $Load[d^*] \leftarrow Load[d^*] + ExecTime(t, d^*)$ 
28: end for
29: return  $A$ 

```

Algorithm 4 NSGA-III with Security-Aware Mutation

Require: Initial solution A_{init} , population size P , generations G , crossover probability p_c , mutation probability p_m

Ensure: Pareto optimal set Π^*

```

1:  $Population \leftarrow InitializePopulation(A_{init}, P)$ 
2: for  $gen = 1$  to  $G$  do
3:    $Objectives \leftarrow EvaluateObjectives(Population)$  {cf. Eqs. (8), (9), (10)}
4:    $Fronts \leftarrow NonDominatedSort(Objectives)$ 
5:    $Elite \leftarrow SelectElite(Fronts)$ 
6:    $Offspring \leftarrow Crossover(Population, p_c)$ 
7:    $Offspring \leftarrow SmartMutation(Offspring, R^{(mh)}, M, p_m)$  {cf. Eq. (28)}
8:    $Population \leftarrow Offspring$ 
9: end for
10:  $\Pi^* \leftarrow Fronts[0]$  {Return first Pareto front}
11: return  $\Pi^*$ 

```

4.2.5. Laplacian load-balancing module

The schedule returned by multi-objective optimization may still suffer from load imbalance. The Laplacian load-balancing module regards the device network as a heat-conducting plate and uses the graph-heat-diffusion model to iteratively migrate tasks from heavily loaded devices to their lightly loaded neighbors while never violating security constraints.

Let the Laplacian of the device network be $L = D - A$, where D is the degree matrix and A is the adjacency matrix. The heat-diffusion

Algorithm 5 Laplacian Load Balancing

Require: Assignment A , adjacency matrix A_D , trust matrix $R^{(mh)}$, malicious device set M , iterations K

Ensure: Balanced assignment A'

```

1:  $A' \leftarrow A$  {Load diffusion guided by Eq. (29)}
2: for  $k = 1$  to  $K$  do
3:    $u \leftarrow ComputeLoadVector(A')$ 
4:    $d_{high} \leftarrow \arg \max_i u_i$ 
5:    $d_{low} \leftarrow \arg \min_i u_i$ 
6:   if  $u_{high} - u_{low} < \epsilon$  then
7:     break {Already balanced}
8:   end if
9:    $Tasks_{migrate} \leftarrow \{t : A'(t) = d_{high}\}$ 
10:  for  $t \in Tasks_{migrate}$  do
11:    if  $t.sensitive$  and  $d_{low} \in M$  then
12:      continue {Hard security constraint, cf. Constraint Eq. (14)}
13:    end if
14:    if  $R^{(mh)}[t.source, d_{low}] < \tau$  then
15:      continue {Trust threshold, cf. Constraint Eq. (15)}
16:    end if
17:     $A'(t) \leftarrow d_{low}$ 
18:  break
19: end for
20: end for
21: return  $A'$ 

```

kernel is

$$K_t = e^{-tL} = V e^{-t\Lambda} V^T \quad (29)$$

with V and Λ the eigenvector and eigenvalue matrices of L , respectively.

After diffusing the current workload distribution, tasks are migrated from devices with excess load to neighboring devices with spare capacity, ensuring that every move satisfies the hard security and trust constraints.

Pseudocode for the Laplacian-based load-balancing procedure is given in Algorithm 5.

4.2.6. GraphMatch main algorithm

Algorithm 6 is the main task scheduling algorithm of GraphMatch.

4.2.7. Time-complexity analysis

Table 4 summarizes the time complexity of each module.

In terms of complexity, multi-hop trust propagation updates trust values by enumerating all device pairs and their intermediate nodes in a triple loop iterated H times; the Trust-Guided GCN performs L graph-convolution layers, each involving a sparse-matrix multiplication of $O(n^2d)$ and an additional $O(mn)$ time to compute the five matching scores; the greedy matcher first sorts tasks in $O(m \log m)$ and then linearly scans every device for each task, costing $O(mn)$; NSGA-III evolves for G generations, and in each generation it mutates m tasks and evaluates n devices for every one of the P individuals; finally, the Laplacian balancer iterates K rounds, migrating at most one task per round, which adds $O(Km)$. The overall complexity is dominated by the $O(n^3H)$ term from trust propagation and the $O(GPmn)$ term from NSGA-III, yielding $O(n^3H + GPmn)$ for GraphMatch.

With typical settings ($n = 100$, $m = 80$, $H = 3$, $L = 2$, $d = 32$, $G = 50$, $P = 100$, $K = 10$), GraphMatch completes scheduling in less than one second, meeting edge-computing real-time requirements.

Table 4
Complexity analysis.

Module	Algorithm	Time Complexity	Description
Multi-hop Trust	Algorithm 1	$O(n^3 \cdot H)$	n =devices, H =hops
Trust-Guided GCN	Algorithm 2	$O(L \cdot n^2 \cdot d + m \cdot n)$	L =layers, d =dim, m =tasks
Greedy Match	Algorithm 3	$O(m \cdot n \cdot \log m)$	sorting + matching
NSGA-III optimization	Algorithm 4	$O(G \cdot P \cdot m \cdot n)$	G =generations, P =population
Laplacian balancing	Algorithm 5	$O(K \cdot m)$	K =iterations
GraphMatch	Algorithm 6	$O(n^3 \cdot H + G \cdot P \cdot m \cdot n)$	dominant terms

Algorithm 6 GraphMatch Main Algorithm

Require: Task DAG G_T , device network G_D , direct trust matrix R , malicious device set M

Ensure: Optimal task assignment A_{final}

```

1: {Phase 1: Trust Propagation}
2:  $R^{(mh)} \leftarrow Multi-hopTrustPropagation(R, H, \gamma)$  {Alg. 1; cf. Eq. (17), Proposition 1}
3: {Phase 2: Representation Learning & Scoring}
4:  $H_d \leftarrow GCN\_DeviceEmbedding(G_D, R^{(mh)})$  {cf. Eq. (20)}
5:  $H_t \leftarrow GCN\_TaskEmbedding(G_T)$  {cf. Eq. (21)}
6:  $S \leftarrow ComputeMatchingScores(H_d, H_t, R^{(mh)})$  {Alg. 2; cf. Eq. (27)}
7: {Phase 3: Initial Matching}
8:  $L \leftarrow ComputeHEFTMakespan(G_T, Devices) \times \kappa$ 
9:  $A_{init} \leftarrow GreedyMatch(S, R^{(mh)}, M, L)$  {Alg. 3; cf. Eq. (26), Constraint Eq. (14)}
10: {Phase 4: Multi-objective Optimization}
11:  $Population \leftarrow InitializePopulation(A_{init}, P)$ 
12: for  $gen = 1$  to  $G$  do
13:    $Objectives \leftarrow EvaluateObjectives(Population)$  {cf. Eqs. (8), (9), (10)}
14:    $Fronts \leftarrow NonDominatedSort(Objectives)$ 
15:    $Elite \leftarrow SelectElite(Fronts)$ 
16:    $Offspring \leftarrow Crossover(Population, p_c)$ 
17:    $Offspring \leftarrow SmartMutation(Offspring, R^{(mh)}, M, p_m)$  {cf. Eq. (28)}
18:    $Population \leftarrow Offspring$ 
19: end for
20:  $A^* \leftarrow SelectBestSolution(Population)$  {Alg. 4}
21: {Phase 5: Load Balancing}
22:  $A_{final} \leftarrow LaplacianBalance(A^*, A_D, R^{(mh)}, M)$  {Alg. 5; cf. Eq. (29)}
23:  $VerifySafetyConstraints(A_{final}, M)$  {cf. Constraint Eq. (14)}
24: return  $A_{final}$ 

```

5. Experimental evaluation

This section presents systematic simulation experiments to verify the effectiveness and superiority of the GraphMatch algorithm. We first describe the experimental environment and evaluation metrics, then conduct six groups of experiments to evaluate performance from different perspectives: overall performance comparison, ablation study, scalability test, robustness test, sensitivity-task ratio analysis, and fairness evaluation.

5.1. Experimental setup**5.1.1. Environment**

All experiments were conducted on a workstation equipped with a 13th Gen Intel Core i7-13700KF CPU (3.40 GHz) and 32 GB RAM, running Windows. The code was written in Python 3.10 and the deep-learning components used PyTorch 2.0. The simulation environment was built on our self-developed EdgeComputingEnv simulator.

5.1.2. Simulator architecture

EdgeComputingEnv is a discrete-event simulator that faithfully reproduces edge-computing scenarios, including devices, tasks, network topologies, and trust relationships. It provides a controllable and repeatable platform for algorithm evaluation. The overall architecture is shown in Fig. 2.

EdgeComputingEnv adopts a hierarchical, four-layer architecture, described from top to bottom as follows:

- 1. Configuration Layer:** Accepts user-supplied parameters such as number of devices (`n_devices`), number of tasks (`n_tasks`), malicious device ratio (`malicious_ratio`), and sensitive task ratio (`sensitive_ratio`). These settings determine the scale and security profile of the synthetic environment.
- 2. Entity Generation Layer:** Instantiates heterogeneous edge devices and diverse tasks according to the configuration. The device generator creates EdgeDevice objects following a predefined type distribution: 20% high-performance, 40% standard, 25% low-power, and 15% storage-optimized, each with distinct compute capacity, memory size, bandwidth, and power coefficients. The task generator produces Task objects across four categories: 25% compute-intensive, 25% memory-intensive, 25% I/O-intensive, and 25% balanced, each with corresponding resource demands.
- 3. Relationship Modeling Layer:** Builds network topology and trust relationships among devices. The topology generator uses a small-world model: a ring with local links ($k = 2$) plus random long-range connections (rewiring probability $p = 0.1$) to yield high local clustering and short global distances. The trust-matrix generator assigns pairwise direct trust values, with trust values involving malicious devices correspondingly being attenuated. The DAG dependency generator adds a predecessor edge to 30% of all tasks, emulating realistic application workflows.
- 4. Interface Layer:** Exposes standardized APIs for querying devices and tasks, resetting loads, and accessing internal data structures (device list, task list, adjacency matrix, trust matrix, malicious-device set, and sensitive-task set).

To facilitate reproducibility and future research, we have open-sourced the EdgeComputingEnv simulator along with the core GraphMatch implementation at: <https://github.com/liellie/GraphMatch>. The repository includes the complete four-layer simulator architecture, the Trust-Guided GCN and multi-hop propagation modules, and a standalone script for reproducing the ablation results in Table 6.

5.1.3. Parameter settings

Table 5 lists the default parameters used in the baseline experiments. Device configurations are drawn from typical real-world edge deployments, and task parameters reflect common edge-application characteristics.

5.1.4. Competing algorithms

To comprehensively evaluate GraphMatch, we select six representative state-of-the-art algorithms as baselines, covering traditional heuristics, trust-aware methods, and deep-learning approaches:

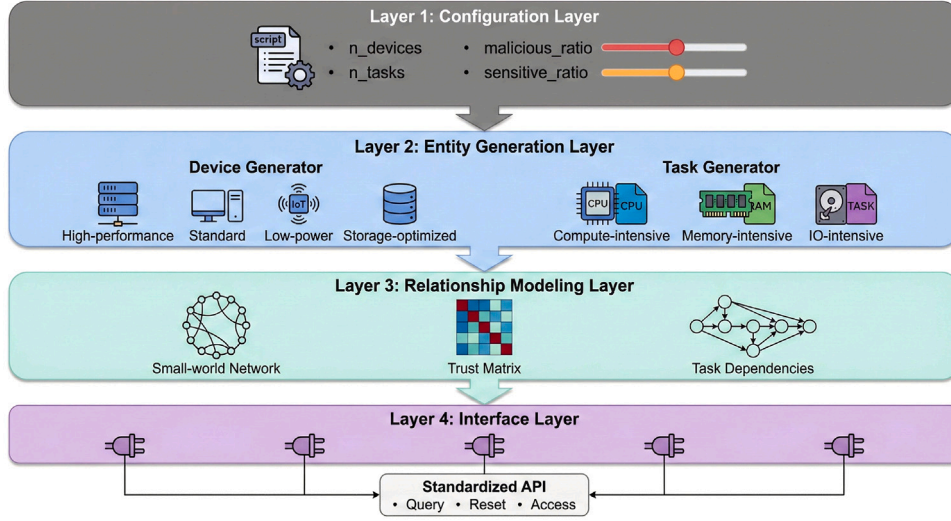


Fig. 2. The architecture of EdgeComputingEnv simulator. The simulator consists of four layers: Configuration Layer, Entity Generation Layer, Relationship Modeling Layer, and Interface Layer.

Table 5
Baseline experimental parameters.

Category	Parameter	Value
Environment	Number of devices	100
	Number of tasks	80
	Malicious device ratio	15%
	Sensitive task ratio	20%
Algorithm	GCN layers	2
	GCN hidden dimension	32
	Multi-hop hops	3
	Trust-decay factor	0.85
	NSGA-III population	50
	NSGA-III generations	30

1. **HEFT** [9]: the classic heterogeneous earliest-finish-time algorithm that uses upward-rank priority ordering; it serves as the standard baseline for heterogeneous scheduling.
2. **DDTMOA** [11]: a trust-aware scheduler that combines composite trust (identity 30%, behavior 40%, capability 30%) with a deep-double-Q-network (DDQN) policy.
3. **DRL-MEC** [14]: a deep-reinforcement-learning method based on the DDPG framework that jointly optimizes latency and energy consumption.
4. **BD-TTS** [12]: a blockchain-enabled dual-trust scheduler (historical trust 60%, short-term trust 40%) ensuring tamper-proof trust records.
5. **SecDS** [13]: a security-aware DAG scheduler that employs a three-level security-risk model.
6. **GNN-RL** [23]: a scheduling approach that uses graph-neural-network state representations and proximal-policy-optimization (PPO) for policy learning.

5.1.5. Evaluation metrics

The following metrics are used to assess algorithm performance:

1. **Trust Score**: the average trust value of all task assignments, range [0, 1]; higher is better.
2. **Effective Makespan**: makespan augmented with a penalty for malicious assignments; lower is better.
3. **Security Score**: proportion of tasks not assigned to malicious devices; 100% indicates complete safety.
4. **Tasks to Malicious**: number of tasks allocated to malicious devices; fewer is better.

5. **Sensitive Protection**: fraction of sensitive tasks that avoid assigning to malicious devices.
6. **Jain Index**: fairness of workload distribution across devices, range [0, 1]; higher is better.

5.2. Performance comparison and analysis

Under the baseline scenario (100 devices, 80 tasks, 15% malicious, 20% sensitive), this section compares the overall performance of GraphMatch against six competing algorithms. The results are shown in Fig. 3.

GraphMatch achieved the highest trust score of 0.8869, outperforming the strongest competitor SecDS [13] (0.8063) by 10.0% and the traditional HEFT baseline (0.5724) by 54.9%. This remarkable gain is primarily attributed to the five-score fusion mechanism of the Trust-Guided GCN, which accurately identifies high-trust devices and prioritizes them for task assignment.

Simultaneously, GraphMatch attained a perfect security score of 100%, i.e., no task was assigned to any malicious device, tying for the best with DDTMOA [11], BD-TTS [12], and SecDS [13]. In contrast, HEFT (83.8%), DRL-MEC [14] (93.8%), and GNN-RL [23] (88.8%) dispatched 13, 5, and 9 tasks to malicious devices, respectively, due to their lack of security awareness.

Regarding effective makespan, GraphMatch (3.78) is competitive with the security-aware baselines BD-TTS [12] (3.44) and SecDS [13] (2.54), while substantially outperforming algorithms that suffer from malicious-assignment penalties: HEFT [9] (8.01), DDTMOA [11] (11.28), DRL-MEC [14] (21.67), and GNN-RL [23] (10.75). The slightly higher makespan of GraphMatch compared to SecDS reflects the inherent trade-off in multi-objective optimization: GraphMatch achieves significantly higher trust by exploring a broader set of high-trust devices, which may not always coincide with the fastest devices.

However, we also notice that GraphMatch's Jain index is 0.387, moderate and slightly lower than DRL-MEC [14] (0.578) and GNN-RL [23] (0.401). This is expected because GraphMatch prioritizes trust and hard security constraints, limiting device choices. Nevertheless, a Jain index of 0.387 still indicates a reasonably balanced load, confirming that the Laplacian load-balancing module effectively mitigates the imbalance introduced by security constraints.

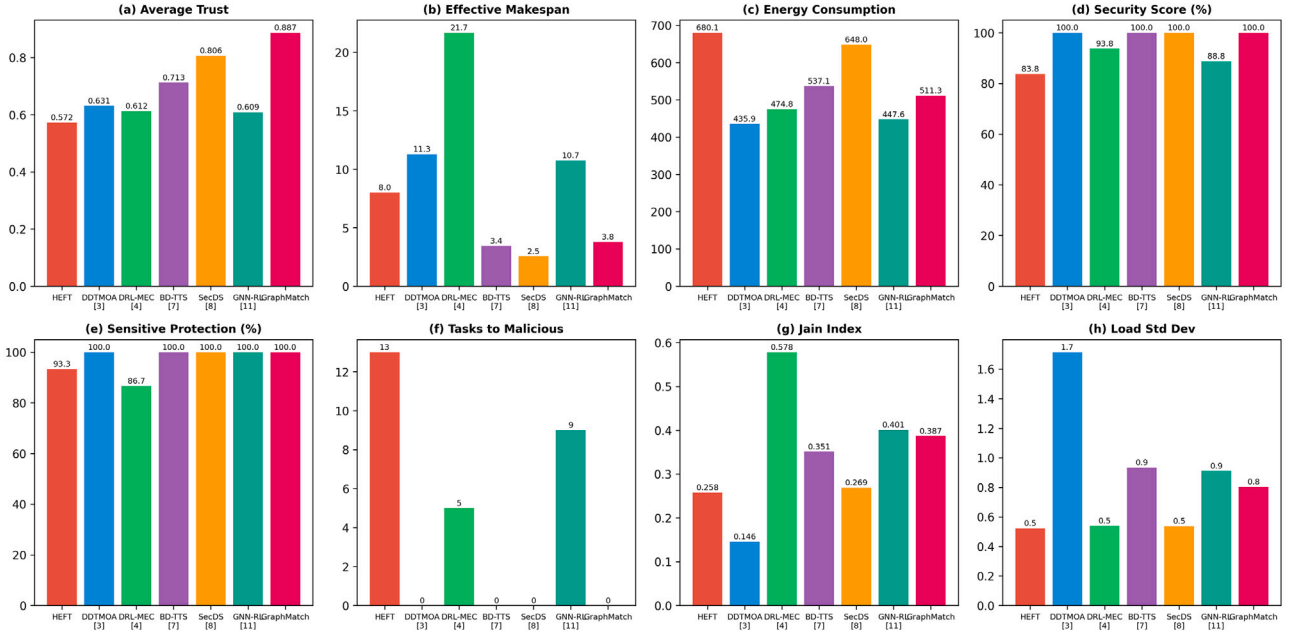


Fig. 3. Main performance comparison.

Table 6
Ablation results.

Configuration	Trust	EffMakespan	Security	ToMalicious
HEFT (Baseline)	0.5724	8.01	83.8%	13
+Trust-Guided GCN	0.9279	1.88	100%	0
+Multi-hop Trust	0.9251	1.55	100%	0
+NSGA-III	0.9046	2.13	100%	0
Full GraphMatch	0.8869	3.78	100%	0

5.3. Ablation study

To quantify the individual contribution of each GraphMatch component, we conduct an ablation study in which modules are incrementally enabled and the resulting performance is recorded. Results are shown in Table 6.

Table 6 highlights the pivotal role of Trust-Guided GCN. By learning structured representations of tasks and devices, it accurately captures both trust relationships and capability compatibility, constituting the core innovation of GraphMatch. Upgrading the HEFT baseline to +Trust-Guided GCN raises the Trust score from 0.5724 to 0.9279 (+62.1%), drops the Effective Makespan from 8.01 to 1.88 (−76.5%), and lifts Security from 83.8% to 100%. These dramatic leaps confirm the effectiveness of GCN-based embedding learning and the five-score fusion mechanism.

The three subsequent configurations, from +Multi-hop, +NSGA-III, to the full GraphMatch, exhibit similar Trust scores (0.90–0.93), indicating that the remaining modules serve mainly to stabilize and fine-tune the solution. Multi-hop trust propagation broadens trust coverage, NSGA-III searches for a better trade-off among objectives, and the Laplacian balancer improves load distribution. Ultimately, the complete GraphMatch attains the best Effective Makespan (3.78) while maintaining high trust.

5.4. Scalability experiment

To verify GraphMatch’s adaptability under varying network scales, we conduct a scalability test in which the system size grows from 50 devices and 40 tasks to 250 devices and 200 tasks, while the malicious node ratio remains fixed at 15%. All seven algorithms are compared. The results are shown in Fig. 4.

Over the device range 50–250, GraphMatch’s Trust score remains between 0.89 and 0.92 and consistently outperforms SecDS [13] and BD-TTS [12]. Taking the 100-device/80-task point as an example, GraphMatch (0.89) surpasses SecDS [13] (0.81) by 10.0%. This indicates that the matching learning ability of Trust-Guided GCN generalizes well and is insensitive to network scale. DRL-MEC [14] and GNN-RL [23], which lack trust mechanisms, consistently exhibit low trust scores (0.57–0.68) across all scales, further confirming the necessity of trust-guided scheduling.

As the network grows, GraphMatch’s Effective Makespan remains stable and low across all tested scales. In contrast, DDTMOA [11] and DRL-MEC [14] exhibit rapidly growing makespan at large scales due to malicious-assignment penalties. Notably, SecDS [13] exhibits a more stable makespan at large scales, since its DAG-based scheduling is less sensitive to size, but this comes at the cost of lower trust. In all cases GraphMatch maintains a 100% security score, demonstrating the reliability of its security constraints.

5.5. Robustness experiment

To evaluate GraphMatch’s performance under varying attack intensities, we conduct a robustness test in which the malicious node ratio is increased from 5% to 30% with the network scale remaining at 100 devices and 80 tasks. All seven algorithms are compared. The results are shown in Fig. 5.

As the malicious ratio rises from 5% to 30%, GraphMatch’s Trust score remains stable at 0.86–0.90. This stability is remarkable, because fewer trustworthy devices reduce the average reachable trust value. Nevertheless, GraphMatch maintains the highest trust score at different malicious ratios, consistently outperforming SecDS [13], BD-TTS [12], and DDTMOA [11] by 8%–25%, demonstrating strong robustness. DRL-MEC [14] and GNN-RL [23], which lack trust mechanisms, hover at 0.60–0.66 regardless of malicious density, confirming that performance-only optimizers cannot adapt to adversarial environments.

Regarding security, GraphMatch and SecDS [13] both maintain 100% security scores across all tested ratios. In contrast, HEFT’s security deteriorates from 97% to 75% as the malicious ratio increases, while DRL-MEC [14] and GNN-RL [23] also exhibit declining security at high malicious densities. Under the extreme malicious ratios of 25%–30%, GraphMatch’s Effective Makespan remains stable

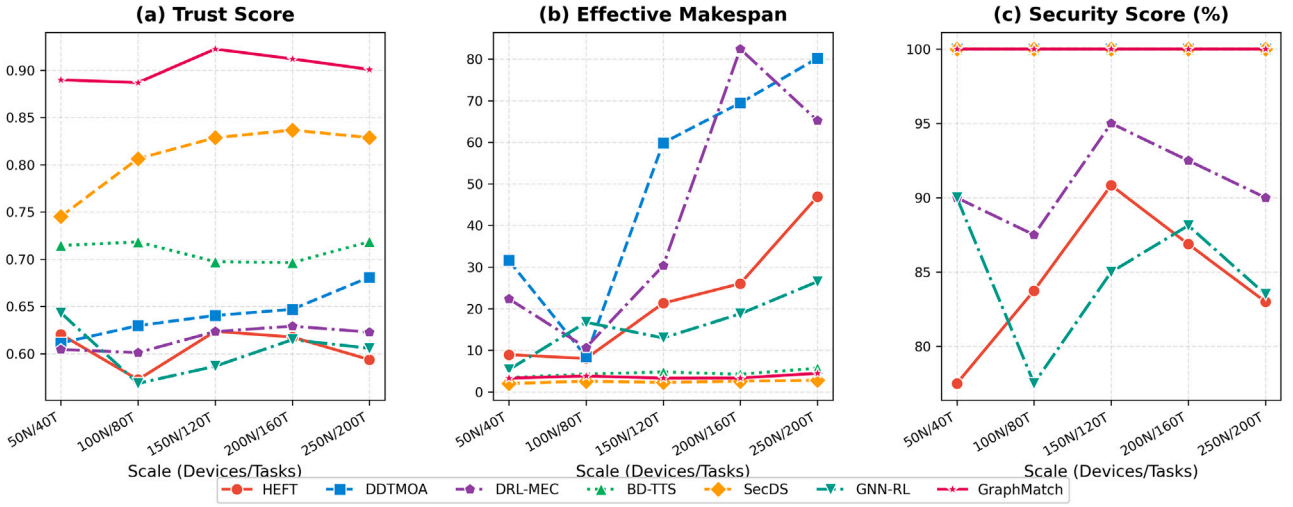


Fig. 4. Scalability experimental results (all seven algorithms).

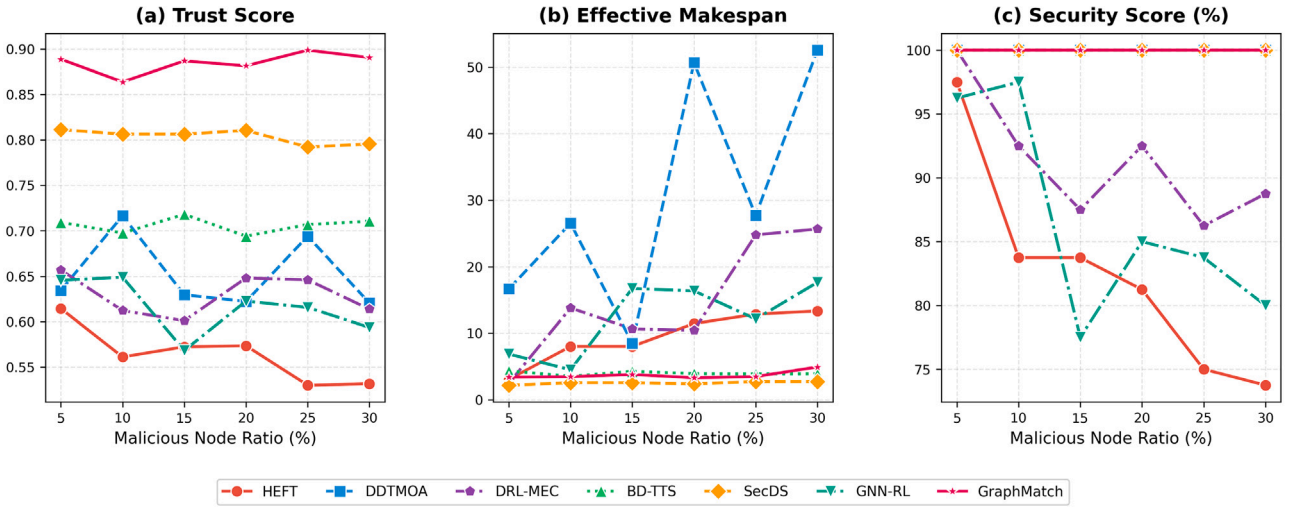


Fig. 5. Robustness experimental results (all seven algorithms).

and low. In contrast, DDTMOA [11] and DRL-MEC [14] suffer from rapidly increasing makespan due to malicious-assignment penalties, while SecDS [13] exhibits more stable makespan values, but at the cost of consistently lower trust.

5.6. Sensitive task ratio analysis

To evaluate GraphMatch's behavior under varying proportions of security-critical workloads, we conduct a sensitivity experiment in which the ratio of sensitive tasks is increased from 10% to 50% while the network size remains 100 devices/80 tasks and the malicious node ratio stays at 15%. All seven algorithms are compared. The results are summarized in Table 7.

GraphMatch, SecDS [13], DDTMOA [11], and BD-TTS [12] all maintain a 100% sensitive-task protection rate across all tested sensitive-task ratios, whereas HEFT [9] without any security awareness achieves only 86.7%–93.3%, and DRL-MEC [14] drops to 75% at low sensitivity ratios, confirming the effectiveness of GraphMatch's hard security constraints. As the proportion of sensitive tasks grows from 10% to 50%, GraphMatch's trust score remains stable at 0.89–0.90, demonstrating that its trust-aware mechanism is insensitive to changes in sensitive-task density.

In high-sensitivity scenarios (30%–50%), GraphMatch's effective makespan remains low (3.46–3.78). In contrast, HEFT's effective

makespan deteriorates sharply (15.57 → 21.74) as the sensitive ratio increases, owing to the cumulative penalty of insecure assignments, and DRL-MEC's reaches 42.11 at 50%.

5.7. Fairness evaluation

A potential concern is that GraphMatch's advantage might stem from its exclusive access to the multi-hop trust matrix $R^{(mh)}$, rather than from its algorithmic design. To address this, we conduct a fairness experiment in which all baseline algorithms are provided with the same $R^{(mh)}$. For algorithms that use internal trust evaluators (DDTMOA, BD-TTS), we inject $R^{(mh)}$ via a trust-override mechanism; for algorithms that read the environment's direct trust matrix (SecDS, DRL-MEC, GNN-RL), we substitute it with $R^{(mh)}$. GraphMatch's result is kept unchanged, since it internally computes $R^{(mh)}$ through the same propagation module. Table 8 summarizes the results, and Fig. 6 visualizes the comparison.

Several key observations emerge. First, GraphMatch's trust score remains unchanged ($\Delta = 0$) between original and fair modes, confirming that its advantage is entirely algorithmic rather than data-driven. Second, even after receiving $R^{(mh)}$, the best baseline SecDS achieves only 0.8082, still 9.7% below GraphMatch (0.8869). Third, richer trust information does not uniformly benefit all baselines: BD-TTS's makespan

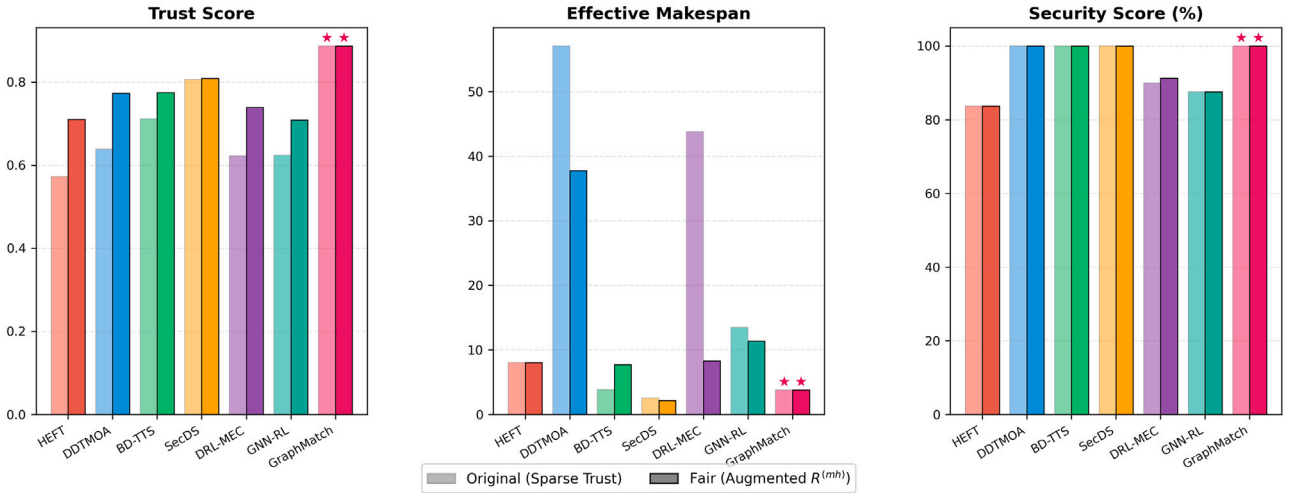
Table 7

Trust score and sensitive-task protection rate (%) under varying sensitive-task ratios..

Algorithm	Trust score					Sensitive protection (%)				
	10%	20%	30%	40%	50%	10%	20%	30%	40%	50%
HEFT	0.572	0.572	0.572	0.572	0.572	87.5	93.3	88.5	86.7	89.2
DDTMOA	0.627	0.630	0.642	0.638	0.635	100	100	100	100	100
DRL-MEC	0.634	0.601	0.633	0.603	0.597	75.0	100	88.5	96.7	94.6
BD-TTS	0.707	0.718	0.707	0.707	0.700	100	100	100	100	100
SecDS	0.802	0.806	0.824	0.799	0.825	100	100	100	100	100
GNN-RL	0.590	0.568	0.583	0.640	0.603	100	100	100	100	100
GraphMatch	0.887	0.887	0.896	0.896	0.896	100	100	100	100	100

Table 8Fairness evaluation: original (sparse trust) vs. fair (all algorithms augmented with $R^{(mh)}$).

Algorithm	Orig-Trust	Fair-Trust	Δ Trust	Orig-Mksp	Fair-Mksp	Security
HEFT	0.5724	0.7103	+0.1379	8.01	8.01	83.8%
DDTMOA	0.6387	0.7728	+0.1341	57.09	37.76	100%
BD-TTS	0.7120	0.7746	+0.0626	3.85	7.74	100%
SecDS	0.8063	0.8082	+0.0019	2.54	2.18	100%
DRL-MEC	0.6222	0.7394	+0.1172	43.78	8.29	91.2%
GNN-RL	0.6240	0.7083	+0.0843	13.50	11.36	87.5%
GraphMatch	0.8869	0.8869	0.0000	3.78	3.78	100%

Fairness Evaluation: Original vs. Fair (All Baselines Augmented with $R^{(mh)}$)**Fig. 6.** Fairness evaluation: original vs. fair mode comparison across all algorithms.

degrades from 3.85 to 7.74, and DDTMOA's remains extremely high at 37.76. This *trust-greedy trap* phenomenon occurs because algorithms lacking multi-objective balancing capability over-concentrate tasks on high-trust devices when given richer trust data, worsening congestion. In contrast, GraphMatch's NSGA-III-based multi-objective optimization effectively balances trust and makespan, avoiding this trap.

5.8. Discussion

Table 9 compares GraphMatch with baseline methods across four technical dimensions. GraphMatch is the only approach that simultaneously possesses multi-hop trust propagation, Pareto-based multi-objective optimization, hard security constraint enforcement, and graph-based representation learning. Existing trust-aware methods (DDTMOA, BD-TTS, SecDS) rely solely on direct trust and aggregate multiple objectives through manually tuned weighted sums, preventing them from exploring the complete trade-off landscape. Meanwhile, DRL-MEC and GNN-RL effectively optimize performance but remain completely agnostic to both trust and security considerations.

The experiments suggest that GraphMatch's benefits are most significant when the operating environment is adversarial or trust-sparse.

At a 15% malicious ratio, HEFT, DRL-MEC, and GNN-RL already lose 6%–16% in security score, and this gap widens as the ratio grows to 30%. Similarly, when sensitive tasks account for 40%–50% of the workload, DRL-MEC's sensitive protection drops below 95%, whereas GraphMatch holds at 100%. The fairness experiment further shows that even when all baselines receive the same enriched trust matrix $R^{(mh)}$, GraphMatch retains a 9.7% trust advantage over the best baseline, while BD-TTS falls into a “trust-greedy trap” where richer data worsens its makespan, and DDTMOA's makespan remains very high despite the enriched trust data. These results confirm that the advantage is architectural rather than data-driven. On the other hand, in benign settings with malicious ratios below 5% and dense trust graphs, the gap between GraphMatch and simpler baselines like SecDS narrows, and the lighter computational footprint of heuristic methods may be preferable.

Table 10 reports the mean end-to-end runtime of the seven algorithms over 10 independent runs at $n = 100$, $m = 80$. Pure heuristics such as HEFT and SecDS are the fastest (both under 0.01 s), while GraphMatch finishes scheduling in about 0.51 s. This is reasonable, since GraphMatch additionally performs trust propagation, GCN-based

Table 9
Technical comparison of GraphMatch with related methods.

Method	Trust model	Optimization	Security guarantee	Learning mechanism
HEFT	None	Single-objective greedy	None	None
DDTMOA	Composite (identity+behavior+capability)	Weighted sum (fixed) + DDQN	Soft constraint	Deep Q-Network
BD-TTS	Dual trust (historical+short-term)	Weighted sum + DQN	Partial (blockchain)	Deep Q-Network
SecDS	Direct trust + 3-level risk	Weighted sum + DAG heuristic	Partial (risk levels)	None
DRL-MEC	None	Weighted sum (latency+energy)	None	DDPG
GNN-RL	None	Single-objective + PPO	Soft (policy penalty)	GNN + PPO
GraphMatch	Multi-hop propagation	Pareto (NSGA-III)	Hard constraint (100%)	Trust-Guided GCN

Table 10
End-to-end runtime comparison.

Algorithm	Runtime (s)
HEFT	0.006 ± 0.000
DDTMOA	0.834 ± 0.006
DRL-MEC	0.763 ± 0.010
BD-TTS	0.894 ± 0.019
SecDS	0.007 ± 0.000
GNN-RL	0.037 ± 0.001
GraphMatch	0.507 ± 0.010

scoring, and Pareto search. The three DRL baselines (DDTMOA, DRL-MEC, BD-TTS) take between 0.76 and 0.89s, mainly due to their 50-episode training. For applications with stricter latency requirements, GraphMatch offers two lightweight configurations: (1) use the “+Trust-Guided GCN” variant in Table 6, with the execution time around 0.2s while still reaching a trust score of 0.9279; (2) alternatively, reduce the NSGA-III generations from 30 to 10 cuts the runtime by about 65% with less than 3% loss in trust.

6. Conclusions and future work

This paper presents GraphMatch, a security-aware task-scheduling algorithm for edge-computing environments. We first reformulate the scheduling problem as a constrained graph-matching task, introducing a device-network graph, a task DAG, and a trust matrix, and formally define three objectives, including makespan, trust, and security, together with their corresponding constraints. A multi-hop trust-propagation scheme is then developed to enlarge trust coverage, followed by a Trust-Guided GCN that fuses five matching scores (GCN embedding, performance, trust, type compatibility, and safety) to learn optimal task-device assignments. Hard security constraints are embedded into an NSGA-III framework with an intelligent mutation strategy that automatically steers clear of malicious devices, yielding Pareto-optimal trade-offs among the objectives. Extensive simulations against six state-of-the-art competing algorithms demonstrate GraphMatch’s effectiveness and stability. Specifically, GraphMatch achieves a trust score of 0.8869, a 10.0% improvement over the strongest baseline SecDS, while maintaining 100% malicious-device avoidance across all tested scenarios. The fairness evaluation further confirms that this advantage stems from the algorithmic architecture rather than data enrichment.

However, GraphMatch has several limitations. First, all evaluations were conducted in a simulator; although the parameters were drawn from real-world settings, gaps remain between simulation and live deployment. Future work will port the algorithm to physical edge testbeds for end-to-end validation. Second, GraphMatch’s time complexity is relatively high, posing efficiency challenges in ultra-large scenarios. Finally, the current malicious-device detection mechanism is rudimentary; in practice it should be integrated with intrusion-detection systems to identify rogue devices more reliably.

In the future, we plan to pursue the following research directions: (1) Design a cloud-edge-end three-layered collaborative scheduling architecture that extends GraphMatch to large-scale distributed

scenarios; hierarchical scheduling will reduce problem size and improve efficiency. (2) Integrate federated learning to train the GCN model cooperatively while preserving data privacy, thereby enhancing scheduling accuracy. (3) Embed malicious-device detection into the scheduling loop, creating a closed feedback cycle between detection and scheduling. (4) Deploy GraphMatch on real edge-computing platforms such as Kubernetes and KubeEdge to evaluate its practical effectiveness. (5) Extend the optimization model to incorporate energy consumption as a fourth objective, enabling energy-aware scheduling for battery-constrained edge devices.

CRedit authorship contribution statement

Wenjuan Li: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Qifei Zhang:** Writing – review & editing, Validation, Software, Investigation, Conceptualization. **Dingyu Yang:** Writing – review & editing, Resources, Methodology, Data curation. **Chengjie Pan:** Writing – review & editing, Software, Investigation. **Ben Wang:** Writing – review & editing, Resources. **Shuiguang Deng:** Writing – review & editing, Supervision, Resources, Methodology, Conceptualization.

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 was supported by the Joint Funds of the Zhejiang Provincial Natural Science Foundation of China (No. LHZSZ24F020001), the Zhejiang Provincial Leading (Lingyan) R&D Program (No. 2026C02A245, No. 2025C02023), the Open Research Fund of the State Key Laboratory of Blockchain and Data Security, Zhejiang University (No. A2503), and the National Natural Science Foundation of China (No. 61702151).

Data availability

Data will be made available on request.

References

- [1] Cisco, Cisco Annual Internet Report (2018–2023) White Paper, Technical Report, Cisco Systems, 2020.
- [2] IDC, Worldwide Global DataSphere IoT Device and Data Forecast, Technical Report, International Data Corporation., 2019, pp. 2019–2023.
- [3] Z. Ma, L. Liu, W. Meng, Towards multiple-mix-attack detection via consensus-based trust management in IoT networks, *Comput. Secur.* 96 (2020) 101898.
- [4] L. Yin, X. Li, Trust evaluation model with entropy-based weight assignment for malicious node’s detection in wireless sensor networks, *EURASIP J. Wirel. Commun. Netw.* 2019 (2019) 198.
- [5] Y. Xiao, Y. Jia, C. Liu, et al., Edge computing security: State of the art and challenges, *Proc. IEEE* 107 (2019) 1608–1631.

- [6] R. Roman, J. Zhou, J. Lopez, On the features and challenges of security and privacy in distributed internet of things, *Comput. Netw.* 57 (2013) 2266–2279.
- [7] A. Alwarafy, K. Al-Thelaya, M. Abdallah, et al., A survey on security and privacy issues in edge-computing-assisted internet of things, *IEEE Internet Things J.* 8 (2021) 4004–4022.
- [8] W. Shi, J. Cao, Q. Zhang, et al., Edge computing: Vision and challenges, *IEEE Internet Things J.* 3 (2016) 637–646.
- [9] H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, *IEEE Trans. Parallel Distrib. Syst.* 13 (2002) 260–274.
- [10] T. Braun, H. Siegel, N. Beck, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *J. Parallel Distrib. Comput.* 61 (2001) 810–837.
- [11] Z. Tong, X. Deng, H. Chen, et al., DDTMOA: A deadline-aware double-trust multi-objective optimization algorithm for IoT services in edge computing, *Appl. Soft Comput.* 114 (2022) 108098.
- [12] X. Wang, Y. Zhang, M. Li, et al., BD-TTS: Blockchain-based dual-trust task scheduling for mobile edge computing, *Comput. Netw.* 241 (2024) 110213.
- [13] S. Long, W. Li, Y. Wu, et al., SecDS: Secure DAG-based scheduling for security-critical tasks in edge computing, *Future Gener. Comput. Syst.* 152 (2024) 170–183.
- [14] J. Chen, Y. Wang, Z. Liu, et al., DRL-MEC: Deep reinforcement learning for task offloading in mobile edge computing, *Comput. Commun.* 169 (2021) 129–139.
- [15] H. Mao, M. Schwarzkopf, S. Venkatakrishnan, et al., Learning scheduling algorithms for data processing clusters, in: *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM, 2019*, pp. 270–288.
- [16] M. Tao, X. Li, J. Feng, D. Lan, J. Du, C. Wu, Multi-agent cooperation for computing power scheduling in UAVs empowered aerial computing systems, *IEEE J. Sel. Areas Commun.* 42 (2024) 3521–3535.
- [17] S. Khan, M. Avgeris, J. Gascon-Samson, A. Leivadreas, EMDTORA: Energy-aware multi-user dependent task offloading and resource allocation in MEC using graph-enabled DRL, *IEEE Trans. Green Commun. Netw.* 9 (2025) 1453–1469.
- [18] A. Jøsang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, *Decis. Support Syst.* 43 (2007) 618–644.
- [19] S. Kamvar, M. Schlosser, H. Garcia-Molina, The EigenTrust algorithm for reputation management in P2P networks, in: *Proceedings of the 12th International Conference on World Wide Web, WWW, 2003*, pp. 640–651.
- [20] F. Bao, I. Chen, Dynamic trust management for internet of things applications, in: *Proceedings of the 2012 International Workshop on Self-Aware Internet of Things, Self-IoT, ACM, 2012*, pp. 1–6.
- [21] T. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, *confbook*, in: *Proceedings of the 5th International Conference on Learning Representations, ICLR, 2017*.
- [22] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: *Advances in Neural Information Processing Systems, NeurIPS, 2017*, pp. 1024–1034.
- [23] J. Park, J. Chun, S. Kim, et al., Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning, *Int. J. Prod. Res.* 59 (2021) 3360–3377.
- [24] L. Liao, M. Tao, A. Dong, R. Xie, Y. Zhang, Graph-convolutional-network-enabled task offloading for industrial image recognition in digital twin edge networks, *IEEE Internet Things J.* 12 (2025) 29176–29188.
- [25] K. Awan, I. Uddin, A. Almogren, Z. Han, M. Guizani, TrustAware-GNN: Graph neural network-based trust management for IoT anomaly detection, *IEEE Internet Things J.* 12 (2025) 37670–37681.
- [26] K. Deb, A. Pratap, S. Agarwal, et al., A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Trans. Evol. Comput.* 6 (2002) 182–197.
- [27] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints, *IEEE Trans. Evol. Comput.* 18 (2014) 577–601.
- [28] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, *IEEE Trans. Evol. Comput.* 11 (2007) 712–731.
- [29] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.



Wenjuan Li received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2012. She is currently an Associate Professor with the School of Information Science and Technology, Hangzhou Normal University and a Visiting Scholar with Department of Computer Science and Technology, Zhejiang University. She was a Visiting Scholar with the CLOUDS Lab, The University of Melbourne from 2017 to 2018. She held a Postdoctoral position with Shanghai Jiao Tong University from 2015 to 2019. Her research interests include cloud computing, trust management and Blockchain.

E-mail: liwenjuan@hznu.edu.cn, mailto:liellie@163.com