Received 12 June 2024; revised 3 July 2024; accepted 8 July 2024. Date of publication 10 July 2024; date of current version 30 July 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3426278

ComputiFi: Latency-Optimized Task Offloading in Multipath Multihop LiFi–WiFi Networks

HANSINI VIJAYARAGHAVAN (Graduate Student Member, IEEE), JÖRG VON MANKOWSKI (Graduate Student Member, IEEE), AND WOLFGANG KELLERER (Senior Member, IEEE)

Chair of Communication Networks, Technical University of Munich, 80333 Munich, Germany

CORRESPONDING AUTHOR: H. VIJAYARAGHAVAN (e-mail: hansini.vijayaraghavan@tum.de)

This work was supported in part by the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the Project "6G Future Lab Bavaria," and in part by the Federal Ministry of Education and Research of Germany (BMBF) in the Programme of "Souverän. Digital. Vernetzt." Joint Project 6G-life.

ABSTRACT The increasing prevalence of latency-critical applications such as Ultra Reliable Low Latency Communications (URLLC), factory automation, and Artificial Intelligence (AI) for image classification demands efficient computational task offloading strategies to meet stringent latency requirements for users. Traditional Wireless-Fidelity (WiFi) networks often fall short due to limitations such as interference and limited bandwidth, necessitating alternative technologies like Light-Fidelity (LiFi), which offer higher data rates and reduced latency. Current single-path offloading approaches do not fully utilize network resources, leading to suboptimal performance. This paper introduces ComputiFi, a task offloading framework that utilizes a multipath, multihop LiFi-WiFi network architecture to minimize latency effectively. By dynamically deciding the offloading destination, splitting data among available technologies, and managing resources across multiple computational entities, ComputiFi addresses the complexities of heterogeneous networks. By capitalizing on multipath transmissions and various potential destinations for offloading, ComputiFi offers a robust solution for scenarios requiring low latency. Employing a variety of optimization tools, including Mixed Integer Nonlinear Programming (MINLP) solvers, meta-heuristics, Deep Reinforcement Learning (DRL), and black-box optimization techniques, ComputiFi processes tasks optimally. Performance evaluations show that ComputiFi consistently reduces user average latency up to 69.3%. Furthermore, for real-time implementation, it offers a DRL solution with prediction time in the order of milliseconds, offering 40.23% improvement in latency over baseline methods.

INDEX TERMS Edge computing, LiFi (light fidelity), multihop, multipath, WiFi (wireless fidelity).

I. INTRODUCTION

IN THE era of rapidly evolving wireless communications, latency-sensitive applications like Ultra Reliable Low Latency Communications (URLLC), factory automation, and Artificial Intelligence (AI) image classification require efficient computational task offloading [1], [2]. These applications demand strict latency and seamless data processing that traditional single-technology networks, particularly Wireless-Fidelity (WiFi), struggle to meet due to interference and limited bandwidth [3]. This creates a need for frameworks that can utilize heterogeneous network resources.

Light-Fidelity (LiFi), a complementary technology to WiFi, is an advantageous solution in indoor environments such as factories where high data rate transmission and low latency are crucial. LiFi operates on the visible light and infrared spectrum, offering inherently safe and Radio Frequency (RF)-interference-free communication, making it ideal for high-density settings where WiFi struggles with bandwidth saturation and security concerns [4]. Additionally, LiFi's capability to provide localized connectivity aligns with the needs of factory automation systems that require precise, reliable, and rapid communication between machinery and control systems.

The integration of LiFi and WiFi ensures that users can offload tasks through an optimal combination of paths, balancing speed, coverage, and reliability [5]. Existing single-path offloading solutions [6], [7] only partially exploit network resources, often increasing latency. This limitation highlights the need for multipath transmission strategies that can leverage multiple concurrent channels to enhance data throughput and reliability [8]. Thus, designing a task offloading solution in multipath, multihop LiFi-WiFi networks presents an opportunity to reduce latency.

Traditional single fixed offloading destinations [9] contribute significantly to network congestion and latency spikes. To address this, multiple levels of Multi-access Edge Computing (MEC) destinations are proposed, including local processing at the user, a nearby LiFi Access Point (AP), a WiFi AP, routers interconnecting these APs, or cloud servers accessible via the network backbone.

This enables a more distributed computation model that brings processing capacity closer to the source of traffic [6], [10], resulting in a multihop infrastructure. The multihop design requires continuous evaluation of routing strategies [11] and allocating resources based on current network conditions, traffic congestion, and the computing load of each destination.

While many existing solutions propose multihop network architectures to extend coverage and enhance connectivity, they often oversimplify latency calculations by merely aggregating latencies across hops, ignoring the complexities of multi-packet data flows and intermediate forwarding hops [6], [12], [13]. Such approaches are inadequate for scenarios where the task's data traffic is composed of multiple data packets.

In response to these challenges, we propose a task offloading framework called ComputiFi, which uses multiple optimization techniques to minimize latency in multipath, multihop LiFi-WiFi networks. Our framework decides the offloading destination, dynamically splits data among available technologies, and allocates computational resources efficiently across the potential compute server destinations.

The framework employs a variety of solvers, including Mixed Integer Nonlinear Programming (MINLP) solvers, meta-heuristics, Deep Reinforcement Learning (DRL), and black-box optimization techniques. This diverse set of tools enables ComputiFi to handle the unique challenges of heterogeneous networks, meeting specific network requirements.

A. CONTRIBUTION

In this paper, we present the architecture and methodology of ComputiFi, demonstrating its potential to significantly reduce the latency of tasks in LiFi-WiFi networks. This paper makes the following contributions:

1) *Task Offloading Framework:* We introduce ComputiFi, a novel framework that offloads latency-critical tasks across multipath, multihop LiFi-WiFi networks. It identifies the optimal offloading destination and effectively splits data flows between LiFi and WiFi, using a multihop latency formulation suited for tasks with multiple packets.
2) *Advanced Optimization Solvers:* We implement multiple optimization tools within ComputiFi, including MINLP solvers (e.g., Gurobi), meta-heuristics, DRL, and black-box optimization techniques. These techniques are designed to adapt to the network's changing conditions and traffic demands, providing robust offloading.
3) *Resource Allocation Strategy across varied Devices:* Our framework implements a dynamic, varied destination strategy that optimally routes tasks across different network devices, including local devices, LiFi APs, WiFi APs, routers, and cloud servers, that change over time.
4) *Benchmarking and Performance Validation:* We benchmark ComputiFi against various baseline solvers, demonstrating its superior performance in latency optimization. The benchmarking results validate the effectiveness of our framework in handling latency-critical applications, making it suitable for emerging use cases in heterogeneous network environments.

B. ORGANIZATION

The structure of this paper is outlined as follows. Section II discusses related research. Section III elaborates on the system models for LiFi and WiFi, task models, MEC processor models, and mobility models for indoor networks. The optimization problem is developed into a mathematical model in Section IV. Section V outlines the solution approaches, including meta-heuristic and DRL-based algorithms. The simulation setup and results are presented in Section VI. The paper concludes with a summary of findings in Section VII and explores future research avenues in Section VIII.

II. RELATED WORK

This section reviews the current literature related to task offloading and resource allocation within mobile and edge computing networks, highlighting key contributions and identifying existing limitations.

A comprehensive survey on computation offloading in edge computing networks is presented in [14]. This work reviews various offloading objectives and methodologies to enhance computational efficiency, identifying key trends, challenges, and open questions. However, it neglects the unique challenges of LiFi-WiFi heterogeneous networks, such as blockages and AP density. While it does mention a few papers that explore multipath routing of tasks, it does not examine this aspect in detail.

Similarly, [15] introduces a mobility-aware framework that coordinates task scheduling and resource allocation in a cooperative device-to-device computing network. While they do not evaluate a multipath network, they do account for user mobility, which makes it relevant for our work where the users are also mobile.

Another work, [16], presents a deep learning-assisted method for online task offloading aimed at minimizing latency in heterogeneous mobile edge settings. Their real-time adaptive technique responds to changing network conditions and user demands, focusing on reducing system latency. While they do not consider a multipath network, their approach to using DRL for real-time implementation and comparing it to the Gurobi solver for optimality is similar to our research process.

Moving towards heterogeneous wireless networks, [6] discusses a framework for task offloading and service caching within WiFi-cellular heterogeneous networks. Their strategy optimizes resource and channel allocation while reducing latency and energy consumption. While they do consider technologies that are diverse in their properties, they do not evaluate multipath transmissions in this scenario.

Reference [8] proposes a novel approach using Genetic Algorithm (GA) and DRL to manage task offloading and resource allocation across multiple Radio Access Technologies (RATs) for URLLC and enhanced Mobile Broadband (eMBB). The model is designed to adapt dynamically to fluctuating network conditions, optimizing for the offloading success rate and maximizing spectral efficiency. Although the authors explore multipath transmission for eMBB tasks, the specifics of using two diverse technologies like LiFi and WiFi warrants new research. Moreover, they do not consider the choice of multiple offloading destinations, which adds a new degree of freedom to the optimization and can reduce the latency further by distributing the tasks through the network.

Taking a step in the direction of multiple offloading destinations, [17] allows for adjacent MEC servers to collaborate to process the tasks. This is not truly a multihop architecture since the task is first sent to the associated server and only forwarded to the adjacent server due to a lack of computational resources, and it does not support multipath transmissions.

Further extending the computing destinations to multiple edge and cloud servers, [10] utilizes multi-agent deep reinforcement learning for cooperative computing offloading and route optimization. They also evaluate the possibility of multiple transmission paths but do not consider users that are mobile and change their wireless associations.

Combining both multipath and multihop aspects in mobile ad-hoc networks, [12] explores task offloading strategies to balance communication delay and energy efficiency through an adaptive path selection algorithm that takes into account the mobility of drones in their network. Their communication model, however, is not entirely suitable for the multi-packet tasks that are relayed across multiple hops.

In summary, unlike existing models that often focus on single-hop or single-path offloading strategies, ComputiFi employs a set of optimization tools, including MINLP solvers, meta-heuristics, DRL, and black-box techniques to dynamically adjust offloading destinations and resource allocations across multiple network layers and wireless technologies. These tools are compared and evaluated for their suitability in task offloading problems. Additionally, while many studies explore reinforcement learning in edge computing, they typically do not address the specific latency and resource allocation challenges in LiFi-WiFi configurations.

III. SYSTEM MODELS

The system model for latency-optimized task offloading in multipath multihop LiFi-WiFi networks is described in the following sub-sections, each targeting a specific aspect of the network model and its operational dynamics. These include the network architecture, the computing server model, the task model, the LiFi and WiFi channel models, and the user mobility model. The notation used in this paper is summarized in Table 1.

A. NETWORK MODEL

This work develops a network model for a LiFi-WiFi network, consisting of M^L LiFi APs and M^W WiFi APs. Each LiFi AP is equipped with Light Emitting Diodes (LEDs) and mounted at a ceiling height of 3 m. Due to operating on the same frequency, these LiFi APs experience co-channel interference within overlapping zones. In contrast, WiFi APs are also positioned at the same height but operate on non-overlapping frequency channels, mitigating inter-cell interference.

The system supports M^U users, each equipped with both LiFi and WiFi transmitters and receivers for uplink and downlink traffic, denoted by $u \in \{1, 2, \ldots, M^U\}$. Connectivity management allows each user to be simultaneously served by one AP of each technology (LiFi and WiFi). However, they do not associate with more than one AP of the same technology. Users are pre-associated with one AP per technology based on Signal to Interference and Noise Ratio (SINR). Therefore, each user has $M^N = 2$ wireless technologies.

APs are interconnected via a router, with one router allocated for every five APs. Each router is directly connected to a cloud server, with backhaul connectivity facilitated through wired links. The network architecture is visually represented in Fig. 1.

The capacity of each link ij in the network is denoted as BW_{ij} and is assumed to be sufficient so that the capacity of backhaul links matches or exceeds the fronthaul bandwidth. Furthermore, the capacity of each router is designed to at least equal the sum of all its links. These capacities are summarized in Table 2, with wireless capacities derived from the respective channel models. Although the propagation delay for most links is negligible due to their proximity in indoor environments, the delay to the cloud server is set to 7.5 ms, as mentioned in existing research [18].

A central network controller links all nodes and manages communication within the network. This controller is critical for collecting Channel State Information (CSI) from users and monitoring server load, facilitating a centralized

TABLE 1. List of notations used in latency-optimized task offloading.

Notation	Description
l, M^L	Index and number of LiFi APs
w, M^W	Index and number of WiFi APs
u, M^U	Index and number of users
l_u	Index of LiFi AP associated to user
w_u	Index of WiFi AP associated to user
n, M^N	Index and number of technologies
m, $\mathcal{M}$	Index, and set of computing destinations
$R_{u,l}, R_{u,w}$	Achievable rate of user for LiFi and WiFi
ij	Link between two nodes i and j in the network
BW_{ij}	Capacity of link ij in bits per second
F_m	Computing server's capacity in instructions per second
F_u	User's processing capacity in instructions per second
t, N_u^T	Type and number of tasks at a user
τ_u^{req}	User's task delay requirement in seconds
C_u	Number of instructions to process user's task
S_u	Data size of user's task in bits
S_u^{pkt}	Size of one packet of the user's task in bits
$\lambda_{ij}^{u,n,m}$	Binary indicator of utilization of links
$x_{u,m}$	Optimization variable for the destination selected for user
k_u^n	Optimization variable for data split proportion
$c_{u,m}$	Optimization variable for compute resource proportion
p_u	Optimization variable to process the task locally or not
τ_u^L	Local computation latency
τ_u^{compute}	Edge computation latency
$\tau_u^{\text{Tx},n}$	Transmission latency over one technology
$\tau_{u,n,m}^{\text{flow}}$	Auxiliary variable for the latency of the bottleneck link
τ_u^{Tx}	Total transmission latency
τ_{ij}^{prop}	Propagation latency for a link ij
τ_u^E	Total latency when the task is offloaded
τ_u	Task completion latency
G	Reward for DRL-based algorithm
V_{τ_u}	Latency Violation
N_u^V	Number of users violating the latency bound

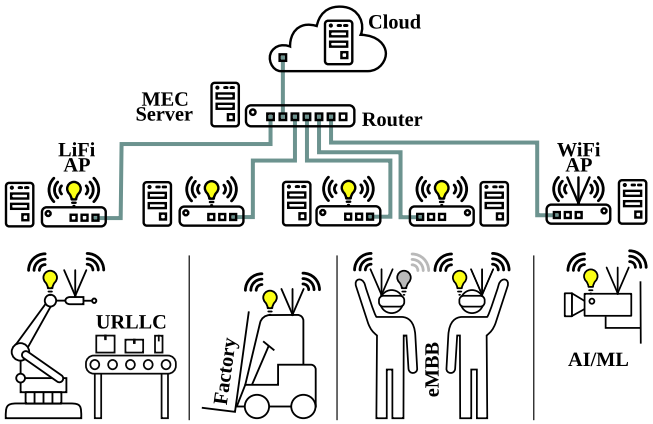


FIGURE 1. Network architecture showing various task applications generated by users associated to LiFi and WiFi APs with other potential offloading destinations equipped with MEC servers.

TABLE 2. Properties of links between network nodes.

From/To	To/From	Capacity (Mb/s)	Propagation Delay (ms)
Router	Router	500	–
Router	Cloud	2500	7.5
Router	WiFi AP	500	–
Router	LiFi AP	500	–
User	LiFi AP	max. 250	–
User	WiFi AP	max. 160	–

TABLE 3. Computational properties of processing servers.

Component	Processing Capacity (Instructions per core per second)	Cores
User	1.9×10^9	1
WiFi AP	5×10^9	8
LiFi AP	5×10^9	8
Router	3×10^9	128
Cloud	3×10^9	128×5

resource allocation algorithm. This algorithm leverages global network information to optimize MEC server destinations, packet split ratios, and computing resource allocation to minimize overall network latency.

B. COMPUTING SERVER MODEL

In the proposed ComputiFi framework, the MEC architecture plays a critical role in reducing latency by providing computational resources closer to the end-users. The processing server destination $m \in \mathcal{M}$ includes cloud servers, routers, and both LiFi and WiFi APs. The user device is also capable of processing but is not equipped with a server. Each component's computational capacity is crucial for optimizing the overall performance of the network. Below, we detail each server component's properties, including the processor's capacity in instructions per second and the number of processing cores. The total processing capacity over all cores combined is F_m. The local user device's capacity is denoted by F_u.

Table 3 presents the computation properties for each type of server entity in the network. The processing capacity of a user device is modeled after the Qualcomm Snapdragon 720G processor. APs for both LiFi and WiFi are equipped with processing capacities as in [19]. Routers use the Ampere Altra Max M128-30 processors, which provide processing capabilities suitable for heavy computational tasks, and a cloud server is modeled by five such processors.

C. TASK MODEL

This section details the model used for tasks that must be processed by the system. Tasks arriving at users can be processed locally or offloaded to a server and then processed. Each user has multiple tasks dependent on each other, which are processed at the same destination. Each

TABLE 4. Task properties for various applications.

App	Arrival Rate (tasks/s)	Size (pkts)	Pkt. Length (bytes)	Processing (Instructions)	Latency (ms)
eMBB	25	33	1500	16.5×10^6	500
URLLC	150	1	32	10 560	1
Factory	100	20	1500	1.25×10^6	100
AI	2	417	1500	569×10^6	500

task is assumed to have a delay requirement less than the optimization interval. Users can employ both LiFi and WiFi links to transmit these tasks to the servers, with the choice of technology potentially varying based on current network conditions and task urgency.

The characteristics of these tasks, including the task arrival rate, delay requirements, and task sizes, can vary significantly across the different applications considered, such as eMBB like Video Streaming, URLLC, Factory Automation, and AI or Image Classification. The arrival rate follows a Poisson distribution with a mean arrival rate in tasks/s as mentioned in Table 4. Other properties of the tasks, such as task size, required latency bound, and required processing instructions for different application scenarios, are also outlined in Table 4. The properties of the eMBB and URLLC tasks are taken from [8], while the Factory Automation properties are from [20] and the AI/Image Classification application is derived from [21]. Furthermore, the delay requirement for a user's task is denoted by τ_u^{req}, the number of instructions required to process the task by C_u, the size of the task in bits by S_u, and the size of one packet of the task in bits by S_u^{pkt}.

D. LIFI CHANNEL MODEL

Our LiFi channel model leverages the three-dimensional positioning of APs and users as well as the orientation of user devices, based on the model in [22]. This spatial configuration is important to determine the channel characteristics, particularly the Line-of-Sight (LoS) channel gain as follows

$$\text{HLoS}_{u,l} = \begin{cases} \frac{(m+1)A_p \chi^2 T_s}{2\pi d_{u,l}^2} \cos\phi_{u,l}^m \cos\theta_{u,l} & \text{if } \theta_{u,l} \le \Theta_f \\ & \& \ \phi_{u,l} \le \Phi_f \\ 0 & \text{elsewhere} \end{cases} \quad (1)$$

Here, m denotes the Lambertian order of the LED, A_p represents the area of the receiver photodiode, χ is the refractive index, T_s the optical filter's gain, $d_{u,l}$ measures the three-dimensional distance between user u and AP l, $\phi_{u,l}$ is the angle of irradiance, and $\theta_{u,l}$ is the angle of incidence. The gain is nonzero only when the irradiance and incidence angles fall within the Field of View (FoV) of both the transmitter (Φ_f) and receiver (Θ_f).

The SINR at the receiver is computed using

$$\text{SINR}_{u,l} = \frac{\left(\text{HLoS}_{u,l} P_l k\right)^2}{\sum_{l' \neq l} \left(\text{HLoS}_{u,l'} P_{l'} k\right)^2 + \text{noise}} \quad (2)$$

TABLE 5. LiFi channel parameters.

Parameter	Notation	Value
Optical filter gain	T_s	1
Refractive index	χ	1
Lambertian order	m	1
Half power beam width	$\theta_{1/2}$	60°
Physical area of the receiver	A_p	1 cm^2
FoV of the receiver	Θ_f	90°
FoV of the transmitter	Φ_f	90°
Optical Power of a LiFi Transmitter	P_l	5 W
Modulation bandwidth of LED	B^L	20 MHz
Noise power spectral density	noise	10^{-21} A^2/Hz
Maximum capacity of LiFi	$R_{\max}^L$	250 Mb/s

Here, P_l is the optical transmission power of the transmitter in watts, and k represents the optical to electrical conversion efficiency.

To calculate the link rate, $R_{u,l}$, for a user u to a LiFi AP l, we employ a modified Shannon formula, as outlined in [23]

$$R_{u,l} = \min\left(B^L \log_2\left(1 + \frac{e}{2\pi}\text{SINR}_{u,l}\right), R_{\max}^L\right) \quad (3)$$

In this formula, B^L is the modulation bandwidth of an LED, and $R_{\max}^L$ indicates the maximum data rate, capped at 250 Mb/s for a LiFi AP.

The parameters for simulating the LiFi channel are detailed in Table 5.

E. BLOCKAGE MODEL FOR LIFI CONNECTIONS

In LiFi networks, users' mobility and device orientation can lead to interruptions in LoS signal transmission due to blockages. To address this, our work incorporates a geometric blockage model, drawing parallels from research on obstructions in mmWave systems [24].

This model considers users as cylindrical barriers, which disrupt the direct LoS paths between ceiling-mounted APs and user devices. The cylindrical model, chosen for its ability to accurately reflect the height and width of a typical user, effectively evaluates signal blockage. This approach applies to any user, including robots or other machines, not just humans. We model users as blockages because their mobile nature presents a more complex challenge for maintaining reliable connections. In contrast, stationary obstacles are not modeled as they can be strategically positioned or removed through careful environment planning, thereby minimizing their impact on signal transmissions. Each cylindrical obstruction is specified with a height of $h = 1.8$ m and a radius of $r = 0.2$ m. Blockage occurs when the direct line between an AP and a user's device intersects with this cylindrical volume. In such cases, the data transmission rate from the affected AP to the user drops to zero.

The blockage model is integral to the channel model, simulating blockages in the LoS of LiFi links. It influences

TABLE 6. WiFi channel parameters.

Parameter	Notation	Value
Small-scale fading gain	h_r	2.46 dB
Frequency of WiFi	f_W	2.45 GHz
Transmission Power of a WiFi device	P_w	0.1 W
Bandwidth of WiFi	B^W	40 MHz
Noise power spectral density	noise	10^{-15} A^2/Hz
Maximum capacity of WiFi	$R^W_{\max}$	160 Mb/s

channel quality and, consequently, the wireless LiFi data rate experienced by the user during uplink task transmission. By incorporating the blockage model, we aim to simulate a realistic channel for users, ensuring an accurate representation of the impact on data rates.

F. WIFI CHANNEL MODEL

Our WiFi network operates under the IEEE 802.11n standard, utilizing a channel bandwidth of 40 MHz. Within this configuration, the maximum throughput ($R^W_{\max}$) achievable by a WiFi AP is 160 Mb/s. Under the assumption of frequency reuse with no interference between WiFi APs, the Signal to Noise Ratio (SNR) at the receiver is modeled as per [22]

$$\mathrm{SNR}_{u,w} = \frac{\left(\frac{1}{d^2_{u,w}} \frac{1}{f^2_W} 10^{14.45} h_r P_w\right)^2}{\text{noise}} \tag{4}$$

Here, $d_{u,w}$ represents the three-dimensional distance between the user u and the AP w, h_r denotes the small scale fading gain (with an average power of 2.46 dB as detailed in [22]), f_W is the carrier frequency, and P_w is the power output of the transmitter in watts.

The link rate for a user u to a WiFi AP w is derived using the Shannon formula

$$R_{u,w} = \min\left(B^W \log_2\left(1 + \mathrm{SNR}_{u,w}\right), R^W_{\max}\right) \tag{5}$$

In this equation, B^W represents the transmission bandwidth. The parameters for simulating the WiFi channel are detailed in Table 6.

G. MOBILITY MODEL

To assess the effectiveness of our task offloading strategy in environments with mobile users, we incorporate the Random Waypoint (RWP) mobility model, which additionally accounts for the changing orientation of user devices. This model applies to any user, including robots, automated systems, or other machines, not just humans.

The height of the user devices over time is modeled using a Gaussian distribution. The mean height is 1.4 m for mobile users when moving and 0.8 m for stationary users. The standard deviation for both scenarios is maintained at 5 cm to accommodate slight variations in device height.

After determining the x and y positions of the user devices through the chosen mobility model, these coordinates are utilized to compute the movement direction. This direction is then used to find the Yaw angle of the device's orientation as detailed in [25]. The Pitch angle is characterized by a truncated Laplace distribution, with an average of 28° for mobile users and 0° for those who are stationary. Similarly, the Roll angle follows a truncated Laplace distribution with a mean of -1.35°. Overall, changes in user orientation over time are modeled as a correlated random process, further elaborated in [25].

IV. TASK OFFLOADING PROBLEM FORMULATION

The ComputiFi framework aims to minimize latency for task offloading in a multipath, multihop LiFi-WiFi network environment.

A. COMMUNICATION MODEL

Once the optimization process selects a destination m for a user u, there are n possible paths to the destination. Here, n represents the wireless technology used, which determines the first hop of the transmission. After deciding on the first hop, no further routing is necessary because the network follows a tree topology rather than a mesh. Consequently, the route to the destination becomes fixed. The links used for transmission in the network, denoted as $\lambda^{u,n,m}_{ij}$, are binary indicators that determine if link ij is used for offloading user u's task over technology n to destination m. The bandwidth allocated to user u on link ij is dependent on the traffic size of the user's tasks and the number of users sharing the link. The capacity of the link is split among the users sharing the link according to their data sizes as given by,

$$\mathrm{BW}^{u,n}_{ij} = \frac{\mathrm{BW}_{ij} k^n_u S_u}{\sum_{u'} \sum_{m'} x_{u',m'} \sum_{n'} \lambda^{n',u',m'}_{ij} k^{n'}_{u'} S_{u'}} \tag{6}$$

where BW_{ij} is the total capacity of the link, k^n_u is the optimization decision variable that indicates the proportion of the data size of the user u that is transmitted over the technology n, S_u is the total data size of the user. The denominator of this equation represents the sum of the data sizes of all users' traffic on that link, where $x_{u',m'}$ is the binary optimization decision variable that indicates if the user u' offloads to destination m'. In this way, the wireless resource itself is not directly an optimization variable; however, the data split proportion is a decision variable that influences the wireless resources allocated to the user.

1) MULTIHOP TRANSMISSION LATENCY OVER ONE TECHNOLOGY

The latency to transmit the user's task $\tau^{\mathrm{Tx},n}_u$ through one technology n from the user u to the destination of offloading m is given by

$$\tau^{\mathrm{Tx},n}_u = \sum_m x_{u,m} \left[\sum_{ij} \lambda^{n,u,m}_{ij} \left(\frac{S^{\mathrm{pkt}}_u}{\mathrm{BW}^{u,n}_{ij}} \right) \right.$$

$$+\max_{ij}\left(\frac{\lambda_{ij}^{n,u,m}\left(k_u^n S_u - S_u^{\text{pkt}}\right)}{\text{BW}_{ij}^{u,n}}\right) + \sum_{ij}\lambda_{ij}^{n,u,m}\tau_{ij}^{\text{prop}}\Bigg] \tag{7}$$

In this equation, the total latency $\tau_u^{\text{Tx},n}$ is composed of three main components:

1) Packet Transmission Latency: The first summation term, $\sum_{ij}\lambda_{ij}^{n,u,m}(\frac{S_u^{\text{pkt}}}{\text{BW}_{ij}^{u,n}})$, represents the latency incurred by transmitting a single packet over all hops from the user to the destination. Each hop contributes to the total latency based on the packet size S_u^{pkt} and the bandwidth $\text{BW}_{ij}^{u,n}$ of the link.
2) Bottleneck Link Latency: The second term, $\max_{ij}(\frac{\lambda_{ij}^{n,u,m}(k_u^n S_u - S_u^{\text{pkt}})}{\text{BW}_{ij}^{u,n}})$, captures the effect of the bottleneck link. The bottleneck link is the link with the lowest bandwidth among all links in the transmission path. This link determines the maximum transmission time for the remaining data packets because it restricts the flow rate of data through the network. Specifically, if the total task size is $k_u^n S_u$ and the size of one packet is S_u^{pkt}, the term $k_u^n S_u - S_u^{\text{pkt}}$ represents the remaining data to be transmitted after the first packet. The division by the bandwidth $\text{BW}_{ij}^{u,n}$ of the bottleneck link then gives the transmission time for this remaining data. Since the bottleneck link is the slowest, the entire transmission process must wait for this link to clear its queue, thereby dictating the overall latency.
3) Propagation Latency: The last term, $\sum_{ij}\lambda_{ij}^{n,u,m}\tau_{ij}^{\text{prop}}$, accounts for the propagation delay across all links used in the transmission.

In our framework, we reserve both bandwidth and computing resources for users, thereby eliminating queuing latency at the task level. At the packet level, queuing can occur at the destination computing server, where the packets wait until all packets of the task have been transmitted before the processing can start. Queuing latency can also be incurred at relay nodes where packets may need to wait for transmission, as considered in the Bottleneck Link Latency. These relay nodes are intermediate points in the network where packets are temporarily stored before being forwarded to the next hop or final destination. At these relay points, we assume a sufficient buffer to handle the incoming traffic, thereby effectively managing the queuing latency. Our approach to queuing is simplified because each user is tied to a single application, allowing us to consider a First Come First Serve (FCFS) queue at the user level rather than at the application level. These considerations are included in the latency formulation wherein the transmission and computing latencies are added together to denote that the processing starts only after the complete transmission, and the Bottleneck Link Latency handles the queuing at relay nodes.

2) TOTAL MULTIPATH TRANSMISSION LATENCY

Once the latency over each technology (or path) is calculated, the total transmission latency τ_u^{Tx} is calculated by the slower of the two paths

$$\tau_u^{\text{Tx}} = \max_n\left(\tau_u^{\text{Tx},n}, n \in \{\text{LiFi, WiFi}\}\right). \tag{8}$$

In this work, we only consider managing resources for uplink traffic since we assume that this data traffic size would be much larger than the result of the processing, which has to be communicated back to the user in the downlink.

B. TASK PROCESSING MODEL

When the task is processed on the user device, the local processing latency τ_u^L is given by

$$\tau_u^L = \frac{C_u}{F_u} \tag{9}$$

where C_u is the instructions required to process the task and F_u is the processing capability of the user's processor in instructions per second.

When the task is processed at the server m, the processing latency τ_u^{compute} is given by

$$\tau_u^{\text{compute}} = \frac{C_u}{\sum_m x_{u,m}c_{u,m}F_m} \tag{10}$$

where C_u is the instructions required to process the task and F_m is the total processing capability of the server in instructions per second, and $c_{u,m}$ is the decision variable that represents the proportion of the total computing resources that is assigned to the user u at the server m.

C. TASK COMPLETION LATENCY

Combining the task transmission and processing latency gives the total task completion latency. If the task is processed locally, then this is directly given by τ_u^L. If the task is processed at the edge (i.e., it is offloaded), the task completion latency at the edge τ_u^E is given by,

$$\tau_u^E = \tau_u^{\text{Tx}} + \tau_u^{\text{compute}} \tag{11}$$

Finally, the total task completion latency τ_u is given by

$$\tau_u = p_u\tau_u^L + (1 - p_u)\tau_u^E \tag{12}$$

where the local latency τ_u^L is selected if the binary decision variable $p_u = 1$ or the edge latency τ_u^E is selected if $p_u = 0$.

D. OPTIMIZATION PROBLEM

The optimization problem for minimizing latency in multipath, multihop networks is given by,

$$\min_{p_u, x_{u,m}, k_u^n, c_{u,m}} \sum_u \tau_u \tag{13}$$

$$\text{subject to} \quad \tau_u <= \tau_u^{\text{req}} \qquad \forall u \tag{14}$$

$$\sum_{m} x_{u,m} = 1 \ \forall u \tag{15}$$

$$\sum_{n} k_u^n = 1 \ \forall u \tag{16}$$

$$\sum_{u} (1 - p_u) x_{u,m} c_{u,m} <= 1 \ \forall m \tag{17}$$

$$p_u \in \{0, 1\} \quad \forall u \tag{18}$$

$$x_{u,m} \in \{0, 1\} \quad \forall u, m \tag{19}$$

$$0 < k_u^n \leq 1 \quad \forall u, n \tag{20}$$

$$0 < c_{u,m} \leq 1 \quad \forall u, m \tag{21}$$

where the objective is to minimize the sum of the users' task completion latency. The decision variable p_u is a binary variable with value 1 if local computing and 0 if edge computing. $x_{u,m}$ is also a binary variable with a value of 1 if the user u's task is offloaded to destination m. The data split proportion is denoted by $k_u^n \in [0, 1]$ and the compute resource proportion is denoted by $c_{u,m} \in [0, 1]$. The constraint (14) denotes the latency upper bound for a task, (15) constrains a user's task to be processed at only one destination, (16) says that the data split proportion over all technologies must sum up to one if offloaded (i.e., all parts of task are transmitted), and (17) is included to not exceed the computation capacity at a compute server destination.

Given the complexity of the MINLP problem, especially with the non-linear nature of the objective and some constraints, and the large network size, sophisticated optimization methods are required.

V. METHODS TO SOLVE THE TASK OFFLOADING PROBLEM

A. BASELINES

In order to benchmark and validate the effectiveness of the ComputiFi framework, several baseline strategies are used to solve the task offloading problem. Each baseline embodies a different approach to task allocation in heterogeneous LiFi-WiFi networks and offers a comparative perspective to evaluate the performance gains achieved by ComputiFi.

1) *Local-Only:* All tasks are processed on the local device, minimizing data transfer latency but potentially causing delays and higher user energy consumption due to limited local resources.
2) *AP-Only (Singlepath):* Tasks are sent to a single AP (LiFi or WiFi) for processing, prioritizing LiFi when available due to lower user density associated to one AP.
3) *AP-Only (Multipath):* Utilizes both LiFi and WiFi by splitting data packets between them, reducing delays and congestion, but requires the entire task to reach the same destination for processing.
4) *Local-First:* Initially processes tasks locally, offloading to an external AP (following AP-Only Singlepath) if local processing would exceed latency limits, balancing local resources and external offloading for optimal latency and reliability.
5) *URLLC-Local (Singlepath):* Prioritizes URLLC tasks for local processing to meet latency requirements, routing other tasks to a single AP as in AP-Only (Singlepath).
6) *URLLC-Local (Multipath):* Processes URLLC tasks locally, while splitting other tasks' transmission between LiFi and WiFi to optimize resource use and meet URLLC latency needs.

B. MINLP OPTIMIZER

MINLP solvers, like Gurobi [26], are key for solving task offloading by using branch-and-bound techniques for discrete and continuous variables. However, for larger networks, MINLP becomes computationally expensive. The branching process grows exponentially with the problem size, which makes MINLP solvers less practical in large, real-world networks. Gurobi is thus used for small scenarios to benchmark learning-based and meta-heuristic approaches in larger cases, ensuring proposed solutions are near-optimal.

The complete optimization formulation is too complex to solve directly, so we simplify the problem by transforming functions to make them as linear or convex as possible. This makes the solution process more manageable and efficient. Recalling (7) and (8), we need to perform max operations. To linearize the max function in (7), we introduce an auxiliary variable $\tau_{u,n,m}^{\text{flow}}$ representing the maximum of the data flow latency out of all hops. Additionally, we add a constraint that this variable should be greater than or equal to the flow latency at every hop. Hence, for a specific (u, n, m) index the constraint is given as

$$\tau_{u,n,m}^{\text{flow}} \geq \left(\frac{\lambda_{ij}^{n,u,m} \left(k_u^n S_u - S_u^{\text{pkt}} \right)}{\text{BW}_{ij}^{u,n}} \right) \quad \forall ij \tag{22}$$

This simplification works due to the minimization objective function, which attempts to decrease the value of $\tau_{u,n,m}^{\text{flow}}$ as much as possible. In contrast, this constraint (22) forces it to be greater than or equal to every hop flow latency value. A similar transformation is applied on (8). Applying these transformations allows us to simplify the problem with exact transformations. Henceforth, the solution using this MINLP solver is called the "Expert" model.

C. META-HEURISTICS

Meta-heuristic algorithms provide efficient task offloading solutions in large networks, balancing quality and computational efficiency. The following meta-heuristic techniques are employed:

1) *Genetic Algorithm (GA):* GA [27] is motivated by the principles of natural selection. It evolves solutions through selection, crossover, and mutation.
2) *Differential Evolution (DE):* DE [28] uses vector differences to improve candidate solutions iteratively.
3) *Pattern Search:* This derivative-free optimization technique [29] iteratively explores the solution space by

evaluating neighboring points and adjusting pattern size dynamically.

4) *Particle Swarm Optimization (PSO):* PSO [30] is a swarm-based algorithm where particles move based on their and neighbors' best positions.
5) *Stochastic Ranking Evolutionary Strategy (SRES):* SRES [31] combines evolutionary principles with a ranking mechanism for constrained optimization problems.

All the meta-heuristics implemented have an initial population size of 100 and are run for a maximum of 100 generations. They can be terminated earlier if they have achieved convergence.

D. BLACK-BOX OPTIMIZERS

In addition to meta-heuristic methods, we also employ black-box optimization techniques, particularly useful for problems with highly complex or unknown internal structures. These methods treat the optimization function as a black box, using statistical learning techniques to model the input-output relationships. Our framework incorporates the following black-box techniques:

1) *Random Forest:* Random Forest [32] is an ensemble learning method that uses multiple decision trees trained on random subsets to improve generalization.
2) *Extra Trees:* Extra Trees [33], short for Extremely Randomized Trees, is another ensemble learning technique similar to Random Forest but uses the entire dataset and randomized split points for greater diversity.

In addition, a simple Greedy approach that samples uniformly within the bounds of the variables and repeats this random selection process for multiple iterations is also proposed.

E. DEEP REINFORCEMENT LEARNING

The task offloading problem, with multiple decision variables and constraints, needs a flexible solution adaptable to network dynamics and varied task requirements. By training an agent through repeated network interactions, DRL learns robust policies that maximize network efficiency and minimize latency violations, balancing optimality with real-time feasibility.

Among DRL algorithms, Proximal Policy Optimization (PPO) [34] stands out for its reliability and stability. PPO limits policy updates to a small range, avoiding large performance drops after learning iterations. Its ability to manage high-dimensional action spaces and efficiently use policy gradients makes it ideal for complex decision-making in multipath, multihop networks.

This section describes the problem formulation, including the state space, action space, and reward function used in the DRL model.

State space $\mathcal{S}$ is a set of features representing the current network status and task characteristics:

$$\mathcal{S} = \{l_u, w_u, R_{u,l}, R_{u,w}, t_u, N_u^T\} \tag{23}$$

where, w_u and l_u represent the WiFi and LiFi APs to which each user is connected, $R_{u,w}$ and $R_{u,l}$ denote the link rates of WiFi and LiFi links for each user, t_u represents the task type (eMBB, URLLC, factory, AI/image classification), N_u^T denotes the number of tasks for each user.

Action space $\mathcal{A}$ defines the possible actions:

$$\mathcal{A} = \{x_u, k_u^W, c_u\} \tag{24}$$

where x_u specifies the computational destination for each user's task. Instead of defining one binary variable or action for every user-destination pair ($x_{u,m}$), we directly output the destination index for each user (x_u), reducing the variable space and allowing for easier learning. This is also why the single destination per user constraint (15) is unnecessary. k_u^W is the proportion of data transmitted via the WiFi link, with the remainder $(1 - k_u^W)$ routed through LiFi or entirely through WiFi if LiFi is unavailable. Therefore, the sum over technologies constraint (16) is not necessary. c_u is the proportion of computing resources allocated to each user at the server. The compute proportion assigned to each user is manually normalized after the action is selected, ensuring that the sum of the proportions assigned at each server is at most 1, thereby eliminating the need for constraint (17). In the discrete model, the proportions are discretized to 20 values, while in the continuous model, they range from 0 to 1.

Reward function G is designed to minimize latency while penalizing latency violations. The latency violation V_{τ_u} is calculated as:

$$V_{\tau_u} = \max\left(0, \tau_u - \tau_u^{\text{req}}\right) \tag{25}$$

where τ_u is the task completion latency for a user and τ_u^{req} is the latency requirement. Each user's reward G_u is based on the inverse of their task completion latency:

$$G_u = \frac{1}{\tau_u} \tag{26}$$

Rewards are then scaled to the range [0, 10]

$$G_u^s = \frac{G_u}{\max(G_u)} \times 10 \tag{27}$$

By scaling the reward, we ensure that the learning process is more stable and efficient and balances exploration and exploitation, ultimately leading to better performance and faster convergence. If there are latency violations, the final reward G is penalized according to the number of violating users:

$$G = -N_u^V \times 10 \tag{28}$$

where N_u^V is the count of users violating latency requirements. If no violations exist, the final reward is the sum of scaled rewards:

$$G = \sum_u G_u^s \tag{29}$$

In summary,

$$\text{Reward} = \begin{cases} -N_u^V \times 10 & \text{if latency violation} \\ \sum_u G_u^s & \text{else} \end{cases} \tag{30}$$

This reward function encourages the agent to minimize latency by selecting optimal destinations and resource allocations while adhering to constraints.

The algorithm works by collecting experiences through the environment using the current policy and then computing the advantages of these experiences using Generalized Advantage Estimation (GAE). PPO updates the policy network by maximizing the expected advantage while ensuring the changes to the policy are within a specified range, controlled by a clipping parameter of 0.2. Additionally, PPO updates a value network to predict the expected return, which is used to compute the advantages. The optimization of both networks is performed using gradient descent with the Adam optimizer. In this implementation, both the actor (policy) and critic (value function) networks use a MultiInputPolicy with a three-layer neural network architecture, having [64, 128, 64] neurons, respectively. The PPO algorithm was implemented using the stable-baselines3 [35] v2.3.0 Python package and the environment was set up with the gymnasium v0.29.1 library. All hyperparameters used in the model training are tuned using the Optuna framework [36]. The model was trained for 2.34×10^5 samples with a learning rate of 0.0001. The summary of the working of the PPO algorithm is described in Algorithm 1.

Algorithm 1 PPO Approach to Find the Optimized Policy and Value Network

Initialize: Policy network π_θ, value network V_ϕ, replay buffer $\mathcal{B}$, learning rate $\alpha = 0.0001$, discount factor $\gamma = 0.9$, PPO clipping parameter $\epsilon = 0.2$, Factor for trade-off of bias vs variance for GAE $\lambda = 0.985$
Input: State space $\mathcal{S}$, action space $\mathcal{A}$, reward G
while not converged **do**
  Reset environment, get initial state s_0
  for each episode **do**
    for each time step t **do**
      Select action $a_t \sim \pi_\theta(s_t)$
      Execute action a_t, observe reward G_t and next state s_{t+1}
      Store (s_t, a_t, G_t, s_{t+1}) in $\mathcal{B}$
    end for
  end for
  Compute advantages $\hat{A}_t$ using GAE:

$$\hat{A}_t = \sum_{l=0}^{T} (\gamma\lambda)^l \delta_{t+l}$$

$$\delta_t = G_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

  Update policy network θ by maximizing:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\Bigg[\min\Bigg(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t,$$

$$\text{clip}\Bigg(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\Bigg)\hat{A}_t\Bigg)\Bigg]$$

  Update value network ϕ by minimizing:

$$L^{\text{value}}(\phi) = \hat{\mathbb{E}}_t\Big[(G_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t))^2\Big]$$

  Perform gradient descent on θ and ϕ using Adam optimizer
  Clear replay buffer $\mathcal{B}$
end while
Return: Optimized policy network π_θ and value network V_ϕ

VI. PERFORMANCE EVALUATIONS

A. EVALUATION METHODS

The proposed ComputiFi framework is assessed through comprehensive simulations implemented in Python 3.10.12.

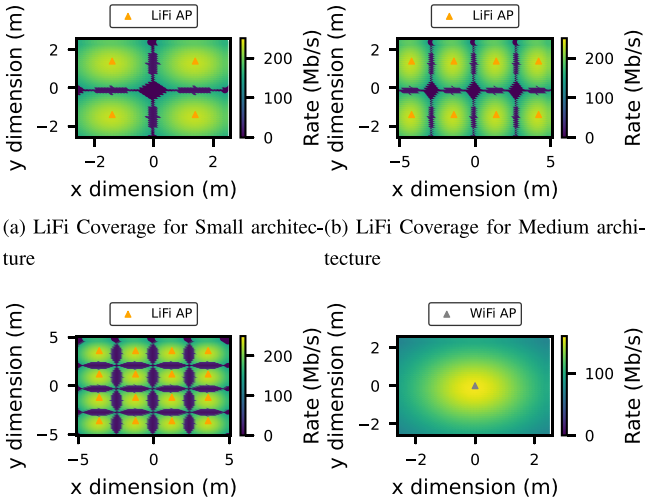
To evaluate the performance of our proposed algorithms, we consider three network architectures: Small, Medium, and Large, which reflect the size of the network topology. In each of these architectures, a single WiFi AP is positioned at the center of the room on the ceiling at coordinates (0,0,3) m. All the LiFi APs are also mounted at the ceiling height of 3 m. The data rate coverage of these network topologies is shown in Fig. 2. The WiFi coverage area is depicted only for the Large topology since the AP is always positioned at the center of the room across all scenarios. Additional parameters for these scenarios are provided in Table 7. Moreover, each architecture includes a Cloud server, and all network devices are equipped with processing capabilities.

The evaluations were conducted using an 11th Gen Intel Core i7-11700 16-Core Processor with NVIDIA GeForce RTX 3070 GPU. All collected results are based on 20 random simulation runs, with each simulation comprising 120 time steps. To verify our claims, we perform hypothesis testing using the Mann-Whitney U test [37], assuming the null hypothesis that the distributions of the two compared parameters are identical. Test results are displayed on the respective figures using [38] and follow the star notation

$$\begin{aligned} \text{ns} &: \text{p} > .05 \\ * &: .01 < \text{p} <= .05 \\ ** &: .001 < \text{p} <= .01 \\ *** &: .0001 < \text{p} <= .001 \\ **** &: \text{p} <= .0001 \end{aligned}$$

Given the multiple hypothesis tests performed on the same dataset, the Benjamini-Hochberg false discovery rate [39] procedure is applied for correction.

Simulation results are evaluated using various quality metrics to provide insights into the offloading framework's performance. For all the latency-related metrics, we compute the task completion latency for each user's task as τ_u.



(a) LiFi Coverage for Small architecture

(b) LiFi Coverage for Medium architecture

(c) LiFi Coverage for Large architecture

(d) WiFi Coverage for Large architecture

FIGURE 2. Data rate coverage achieved for various network architectures under evaluation.

TABLE 7. Task offloading architectures under evaluation.

Scenario	Size	Users	LiFi APs	(x,y) Coordinates LiFi AP	Routers
Small	5 x 5 x 3 m	10	4	(-1.4, 1.4), (1.4,-1.4), (-1.4,-1.4), (1.4, 1.4)	1
Medium	10 x 5 x 3 m	15	8	(-1.4, 1.4), (1.4,-1.4), (-1.4,-1.4), (1.4, 1.4), (-1.4, 4.2), (1.4,-4.2), (-1.4,-4.2), (1.4, 4.2)	2
Large	10 x 10 x 3 m	20	16	(-3.6,-3.6), (-3.6,-1.2), (-3.6, 1.2), (-3.6, 3.6), (-1.2,-3.6), (-1.2,-1.2), (-1.2, 1.2), (-1.2, 3.6), (1.2,-3.6), (1.2,-1.2), (1.2, 1.2), (1.2, 3.6), (3.6,-3.6), (3.6,-1.2), (3.6, 1.2), (3.6, 3.6)	4

Task Latency (ms):

$$\text{Task Latency} = \frac{1}{M^U} \sum_{u=1}^{M^U} \tau_u \tag{31}$$

Latency Quality of Service (QoS): The Latency QoS metric is calculated by taking the ratio of the required latency to the actual latency for each user and then averaging these ratios across all users. This provides a measure of how well the system adheres to the latency requirements of its users.

$$\text{Latency QoS} = \frac{1}{M^U} \sum_{u=1}^{M^U} \frac{\tau_u^{\text{req}}}{\tau_u} \tag{32}$$

A higher Latency QoS value indicates better performance, as it means that the actual latencies are closer to or even lower than the required latencies, ensuring that the tasks are completed within the acceptable time frames. Conversely, a lower Latency QoS value suggests that the system is struggling to meet the latency requirements, resulting in delays that may affect user experience.

Latency QoS per application: The QoS per task application type is calculated by grouping the users according to their application. For example for the URLLC application,

$$\text{Latency QoS}^{\text{URLLC}} = \frac{\tau_{u'}^{\text{req}}}{\tau_{u'}} \qquad \forall u' \in \mathcal{U}^{\text{URLLC}} \tag{33}$$

Further, we present metrics that provide insights into the workings of the solving algorithms.

MEC destination: Since we propose using varied potential destinations for offloading, we examine the MEC server to which each user's task is offloaded.

Proportion of data flow: To understand the benefits of using LiFi and WiFi combined in a multipath network, we analyze the fraction of the data flow of each user that is offloaded through each technology. This value ranges from 0 to 1. This only includes the data for the tasks that are offloaded and the zeros resulting from local processing are not part of the data.

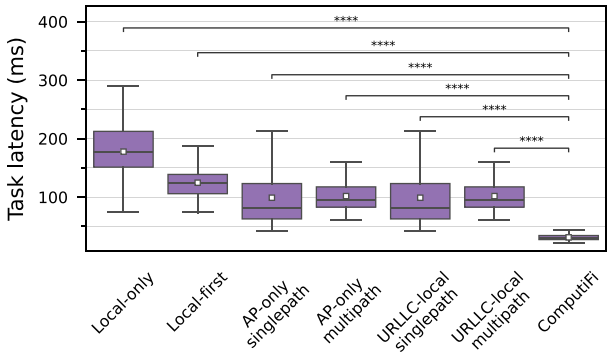
Time to solve (s): Finally, to evaluate the solving algorithm itself, we assess the time needed to solve the optimization problem using the proposed solvers.

B. RESULTS AND COMPARATIVE ANALYSIS

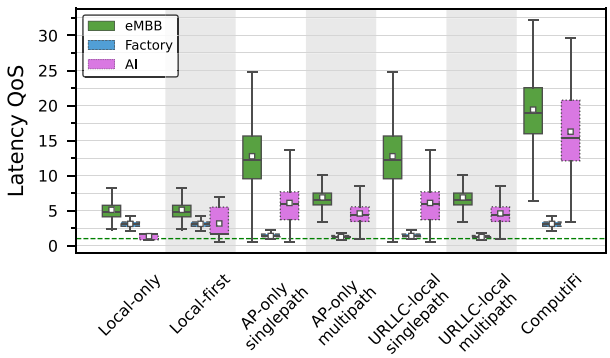
The evaluation begins with Fig. 3(a), which compares various baseline approaches and the proposed solution utilizing the MINLP solver. In this small network topology, each of the 10 users is randomly assigned one of four applications: eMBB, URLLC, Factory, and AI. Due to the random allocation and limited user count, this scenario only involves eMBB, Factory, and AI applications.

As seen in Fig. 3(a), the Expert optimizer achieves the lowest latency for user tasks among all methods, showing the efficacy of our proposed approach. It achieves 69.3% lower average latency than the best baseline method. The inclusion of statistical annotations highlights significant differences between ComputiFi and other methods, proving the value of optimization in computational task offloading.

Further analysis reveals that the Local-Only approach exhibits high latency with considerable variance, reflecting the limitations of processing all tasks locally. The Local-First approach slightly reduces this latency by offloading tasks only when local processing violates latency requirements. Comparing the singlepath and multipath strategies for both AP-only and URLLC-local, we notice a slight reduction in average latency with the singlepath approach while the upper bound is still higher than the multipath approach. This suggests that the multipath transmission reduces the latency, but it is not always the case since a portion of the data flow has to pass through the backhaul back to the destination AP, incurring an extra delay. Furthermore, in this case, both the URLLC-local and the AP-only approaches are the same due to the lack of URLLC application users.

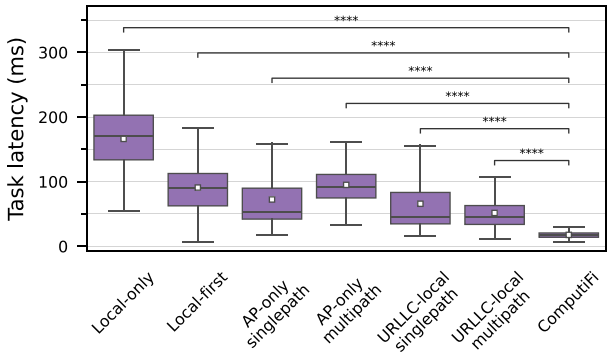


(a) Task Completion Latency

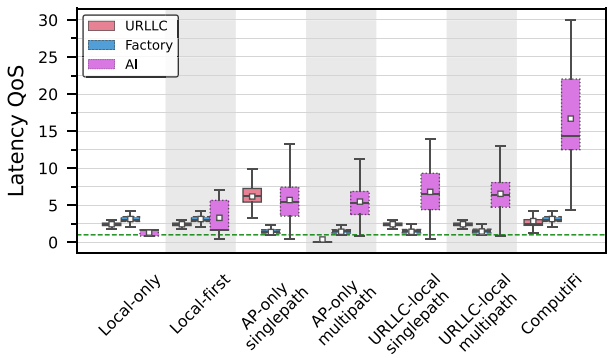


(b) Latency QoS per application where a QoS of 1 (green dotted line) indicates user's latency requirement is exactly satisfied

FIGURE 3. Network quality metrics for the baseline algorithms and ComputiFi's optimized task offloading solved with the Expert for the Small scenario with eMBB, Factory, and AI applications showing the need for our proposed optimization.



(a) Task Completion Latency



(b) Latency QoS per application where a QoS of 1 (green dotted line) indicates user's latency requirement is exactly satisfied

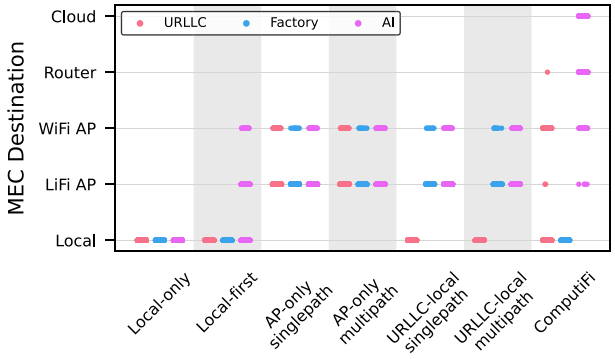
FIGURE 4. Network quality metrics for the baseline algorithms and ComputiFi's optimized task offloading solved with the Expert for the Small scenario with URLLC, Factory, and AI applications showing the efficacy of our proposed optimization for various application scenarios.

After evaluating the overall performance of different baseline strategies and the proposed Expert solution in reducing task latency, we perform a per-application analysis by examining the QoS metric. Fig. 3(b) provides detailed insights into the latency performance for three specific application types: eMBB, Factory, and AI. A QoS value of 1 or higher indicates that the latency requirements are met, and the green line across the plot represents this threshold. For all applications, the Expert optimizer consistently achieves a QoS value above 1, meaning it meets the latency requirements. In general, the latency requirements for eMBB application are always satisfied even with the baseline approaches due to its relatively lenient bound of 500 ms. For the Factory application, most baselines fail to reach the satisfaction level of 1. For the AI application, due to its large task size, baseline approaches often struggle to meet latency requirements consistently. The Expert optimizer reliably maintains a QoS value above 1 for all application types.

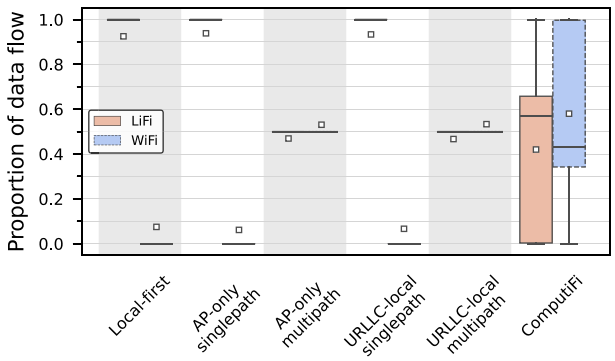
In order to confirm these findings for a network with the stricter URLLC application as well, we randomly generate tasks for users only out of URLLC, Factory, and AI, and the results are visualized in Fig. 4. The optimized solution, once again, achieves the lowest latency as seen in Fig. 4(a), with 65.8% lower average latency than the best baseline (URLLC-local multipath).

Interestingly, multipath transmission provides significant benefits for the URLLC-local method compared to the AP-only approach. This advantage is better explained when analyzing the latency QoS per application. The local processing of the strictly constrained URLLC application alleviates the congestion on the wireless links, allowing multipath transmission to significantly benefit the URLLC-local method. This also results in complete user satisfaction in almost all cases.

A detailed analysis of these algorithms' behavior under the URLLC-focused scenario is provided in Fig. 5. The destinations of task processing selected by the algorithms are visualized in Fig. 5(a). The AI application, being the largest task, is always offloaded except in the Local-only algorithm. The optimized solution effectively utilizes all groups of MEC servers to process AI tasks, significantly reducing latency. Interestingly, the optimized solution offloads URLLC tasks further away, illustrating that multipath LiFi-WiFi networks can reliably deliver ultra-low latency. The Factory task is entirely processed locally by the Expert algorithm, which



(a) MEC destination to which each task is offloaded



(b) Proportion of the traffic of each task transmitted with each technology

FIGURE 5. A detailed look into the working of the baseline algorithms and optimized task offloading solved with the Expert for the Small scenario with URLLC, Factory, and AI applications.

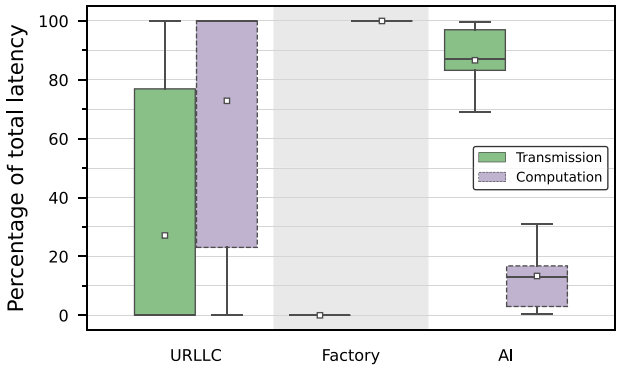


FIGURE 6. Percentage of the individual components of latency per application using ComputiFi's optimized task offloading solved with the Expert for the Small scenario with URLLC, Factory, and AI applications.

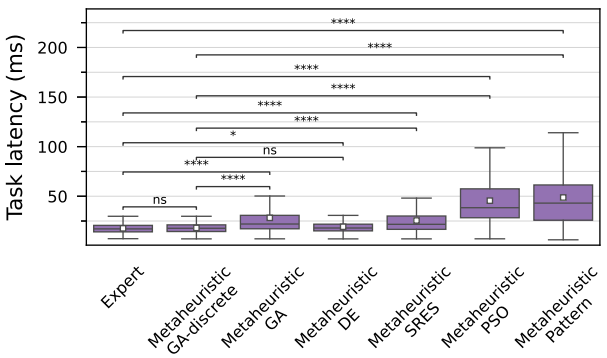
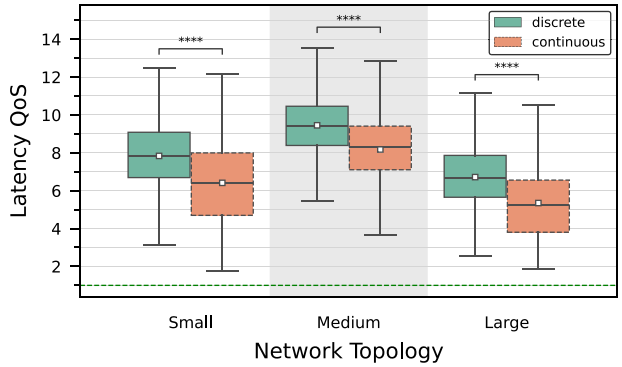


FIGURE 7. Task completion latency achieved by all the meta-heuristic algorithms under test in comparison with the optimal Expert solution for the Small scenario with URLLC, Factory, and AI applications.

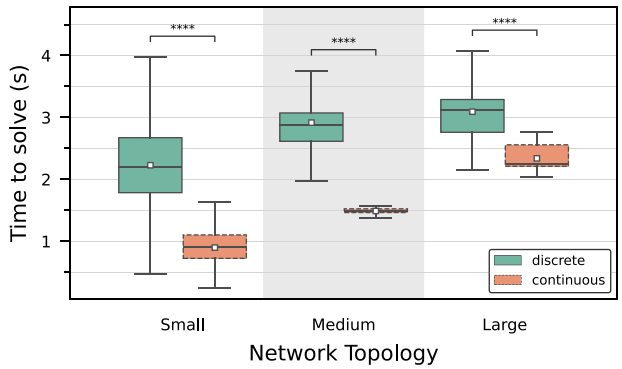
might seem counter-intuitive due to the URLLC tasks being offloaded. Theoretically, both URLLC and Factory tasks can be offloaded and still meet the latency bound. However, the objective of the optimization problem is not just to meet the latency requirement but to minimize it as much as possible. The URLLC task has a 1 ms requirement, which is already low, so further minimization is not feasible. In contrast, the Factory task's 100 ms requirement allows for saving several tens of milliseconds, reducing latency to 20-30 ms and significantly lowering the network sum delay. Hence, our framework processes Factory tasks locally to save significant time and offloads some URLLC tasks to keep them close to but below the 1 ms bound. Another factor leading to this is the size of the data stream. The URLLC tasks consist of a single 32 byte packet, which can be offloaded fast without congesting the link. In contrast, Factory tasks consist of 20 packets of 1500 bytes each, making them slower to offload. This, combined with a higher packet arrival rate, necessitates processing Factory tasks locally for optimal performance.

The usage of multiple paths for offloading is evidenced by the proportion of data flow shown in Fig. 5(b). The baseline multipath algorithms always split the data equally between available technologies. Due to the possibility of blockages in LiFi, the proportion of data flow through it is not always exactly 0.5. The optimized solution balances the use of both LiFi and WiFi networks. The median transmission is higher for LiFi due to its higher data rate, but the average is higher for WiFi due to some samples with LiFi blockages and when the offered rate through LiFi is low. This approach ensures stability and consistently low latency by using multipath offloading.

Fig. 6 presents an analysis of latency components, distinguishing between transmission and computation latencies within a URLLC, Factory, and AI context. It highlights the variability and distribution of these latencies. Transmission latency for URLLC ranges from extremely low percentages indicative of local processing with the median at 0%. The highest value is 99.7%, which indicates efficient offloading to more distant locations with higher computing capacity. In the Factory application, which is processed entirely locally, computation latency accounts for the entirety of the delay. The AI application is almost entirely offloaded, which is evident in the lower percentage of computing latency. The figure shows that for real-time applications, both latency components must be minimized to achieve an optimal balance. This balance between minimizing transmission and computation latency is precisely what ComputiFi is able to provide.

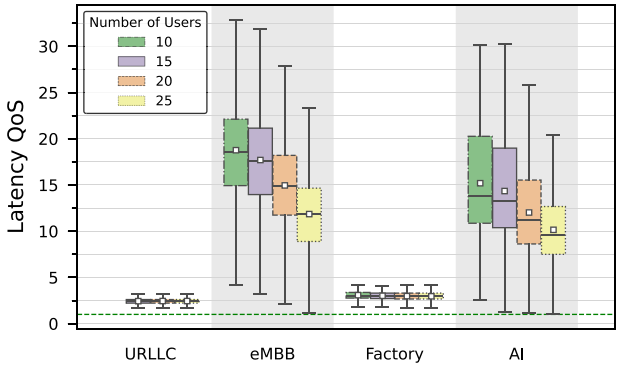


(a) Latency QoS

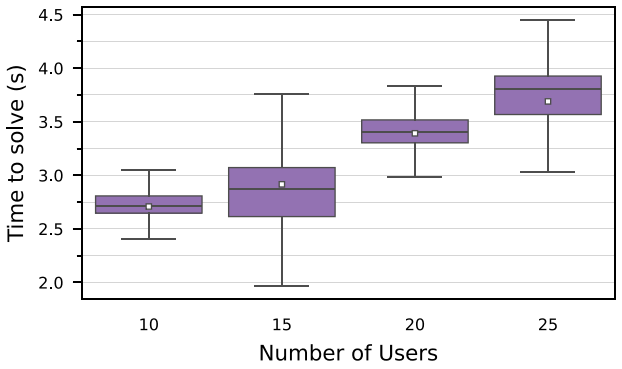


(b) Time to solve

FIGURE 8. Quality metrics for various network architectures using the discrete and continuous GA showing the superiority of the model with discrete variables.



(a) Latency QoS per application



(b) Time to solve

FIGURE 9. Quality metrics for increasing number of users using the discrete GA in a Medium architecture.

While we have established the benefits of our optimized approach to task offloading, this method is not practical for real-time implementation due to its run-time, which is in the order of a few minutes. Therefore, we explore the usage of meta-heuristics and compare their performance to the optimal solver-based results in Fig. 7. The discrete model of the GA and the DE solutions perform exceedingly well and close to optimal. However, on further examination, we confirmed that there is a small yet statistically significant difference between the GA and DE solvers in the QoS metric. As a result, we adopt the GA as the most effective solution method and visualize the results of significance tests, comparing it to all other algorithms. This analysis indicates that the difference between the optimal solver and the GA-based solution is statistically insignificant.

Encouraged by the superior performance of the GA algorithm, we examine its effectiveness across different network architectures, confirming that the discrete model outperforms the continuous one, as shown in Fig. 8(a). However, both models consistently meet all user requirements across all network architectures. Despite the discrete model's superior performance, the continuous model may still be preferable in scenarios where faster solving times are needed, given that it requires less computation time.

To validate the scalability of the proposed GA under increasing user count, we vary the number of users while keeping the number of offloading destinations constant within the Medium network architecture. The results are presented in Fig. 9. When there are just 10 users, the URLLC application is not assigned due to a random selection of applications. For the URLLC and Factory applications, the QoS remains fairly constant while increasing the number of users as seen in Fig. 9(a). This indicates that the algorithm efficiently manages the additional load without compromising the performance of these applications. Conversely, the eMBB and AI applications exhibit a decreasing trend in QoS as the number of users increases. This decline reflects the growing strain on network resources; however, even with up to 25 users, the system still fulfills all user requirements. Further, upon examining the solve time in Fig. 9(b), we observe a linearly increasing trend in the mean values. The increasing time is due to the additional variables and constraints added, which makes the optimization problem larger, and the fact that the framework now has to accommodate more user demands with the same network infrastructure. The linear trend suggests that the GA's performance scales well with the number of users, displaying its potential for deployment in demanding network environments. However,

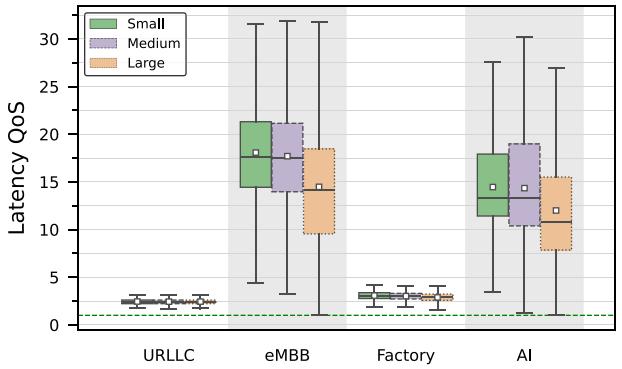


FIGURE 10. Latency QoS per application for varying the number of compute destinations while keeping the number of users fixed at 15 using the discrete GA.

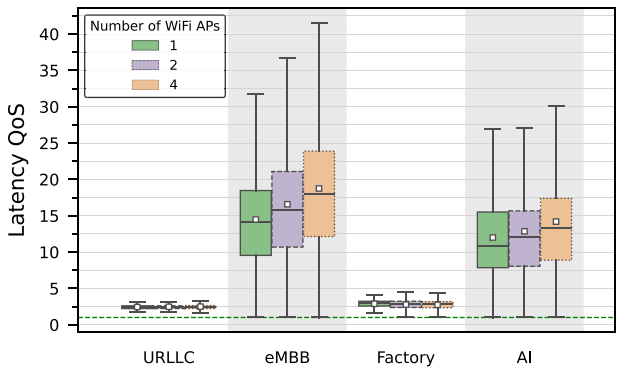


FIGURE 11. Latency QoS per application for varying the number of WiFi APs while keeping the number of users fixed at 15 using the discrete GA.

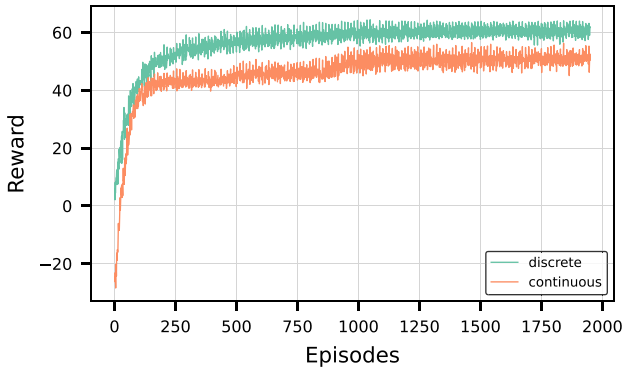


FIGURE 12. The reward obtained during the training episodes of the DRL-based PPO algorithm with discrete and continuous action space in the Small scenario with eMBB, Factory, and AI applications.

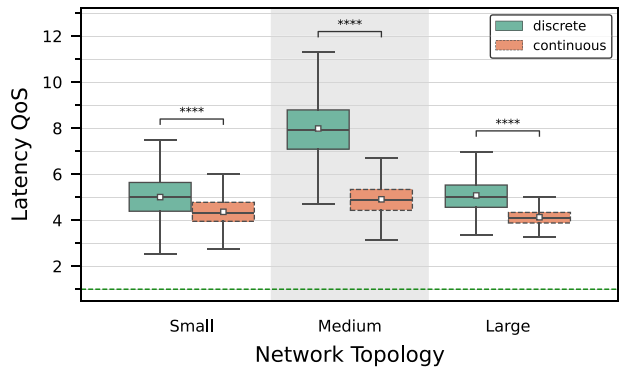


FIGURE 13. Latency QoS for various network architectures using the DRL-based algorithm with discrete and continuous actions.

it also implies that very large numbers of users could lead to significantly larger solve times, necessitating more efficient algorithms.

To validate the findings of ComputiFi with an increasing number of computing destinations while fixing the number of users, we perform this evaluation with the number of users fixed to 15 and visualize the results in Fig. 10. The results indicate that latency increases with the number of LiFi APs, despite the number of users and tasks remaining constant. This increase in latency can be attributed to several factors. In order to be able to increase the number of computing destinations while still reducing interference to allow a fair comparison, we also increase the size of the indoor environment. However, this also means that the distance between the user and a WiFi AP increases, resulting in higher transmission latency, especially when the LiFi signal is blocked. Additionally, more APs increases the likelihood of a user being in the interference region of LiFi APs.

Unlike the LiFi APs, there is no interference among the WiFi APs, so we investigate the effect of increasing the number of WiFi APs while keeping the number of users fixed at 15 in Fig. 11. As expected, we see that the QoS increases with increasing number of APs. This is because there are more potential offloading destinations, reducing latency. Additionally, frequency re-use prevents interference and therefore does not affect the users in the overlapping regions of the APs.

While we have shown that the meta-heuristic GA is scalable, its run time is still in the order of a few seconds, which remains impractical for an optimization interval of 500 ms. This motivates us to explore using a DRL-based approach to address the task offloading problem more efficiently. First, we investigate the convergence of the training process and visualize the reward in Fig. 12 plotted versus the training episodes. Our comparative analysis between the discrete and continuous models within the DRL framework reveals that the discrete model converges to a higher reward value.

After confirming the convergence of the trained model, we apply the cross-validated model on the test set and investigate the results for all network architectures in Fig. 13. Consistent with the reward outcomes observed during training, the discrete model outperforms the continuous one in terms of QoS across all network configurations. This aligns with the GA results and confirms the efficacy of the discrete model in managing task offloading more effectively. We also see that the DRL-based method consistently serves the user with the required QoS.

We also validate that the DRL-based algorithm indeed still performs better than the baselines in Fig. 14. The DRL-based solution, as expected, achieves the lowest latency

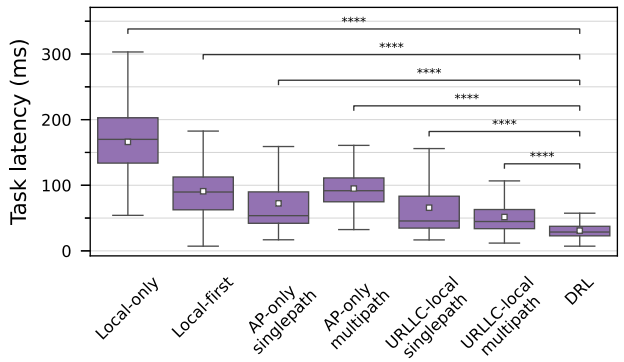


FIGURE 14. Latency for the URLLC, Factory, and AI applications comparing the DRL-based algorithm with the baseline algorithms.

with 40.23% lower average latency than the best baseline (URLLC-local multipath).

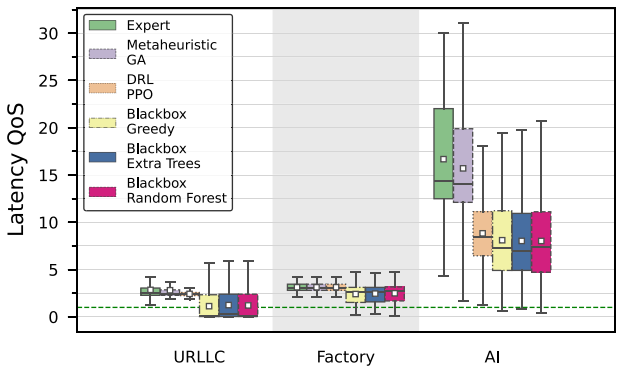
The time to solve for a DRL-based algorithm indicates the time taken to predict a set of offloading decisions from the time the inputs are provided. The results ranging between 7 ms and 12 ms for the Small-Large topology indicate the feasibility of implementing this algorithm on real hardware. The DRL approach not only makes timely decisions but also ensures that these decisions comply with the QoS requirements, demonstrating its suitability for mobile and time-sensitive network environments.

To summarize all investigated optimization algorithms in detail, we directly compare them in Fig. 15. This comparison includes the evaluation of Blackbox algorithms, which have not been discussed previously. These algorithms are particularly important as they help assess whether it is necessary to understand the properties of the problem or function under test.

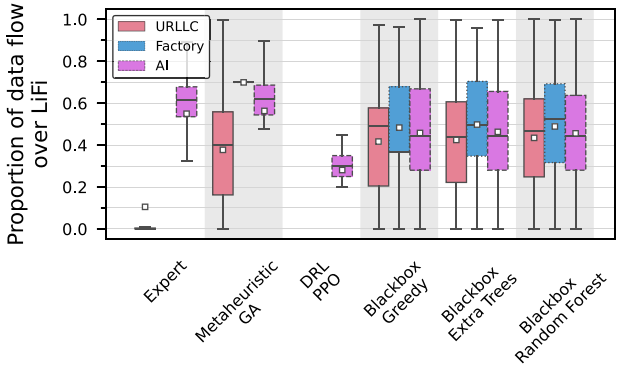
The QoS results in Fig. 15(a) clearly reveal that treating the optimization problem/objective function as a blackbox does not yield near-optimal results. Moreover, except for the simplistic greedy algorithm, Blackbox approaches generally require a significantly longer time to solve the problem. This insight shows the value of using tailored algorithms, like the GA or DRL-based methods, which not only meet QoS requirements more effectively but also operate within acceptable time constraints.

The top three algorithms in terms of network performance, as seen in Fig. 15(a), are the MINLP solver-based Expert solution, the meta-heuristic GA, and the DRL-based solutions. For the applications of URLLC and Factory, these three algorithms show remarkably similar performance levels. However, the DRL solution shows a drop of 47.03% in QoS for the AI application compared to the other two, though it still manages to meet user requirements in all cases.

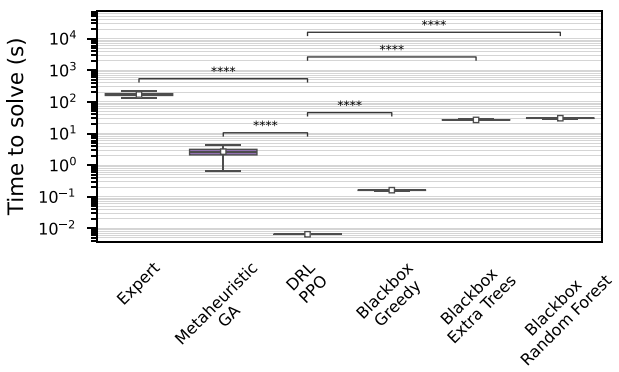
To gain insight into the workings of the different approaches, we visualize the proportion of data transmitted over LiFi (if offloaded), for the URLLC, Factory, and AI applications in Fig. 15(b). The WiFi proportion is the inverse of the LiFi proportion. An interesting observation is that the



(a) Latency QoS per application



(b) Proportion of the traffic of each task transmitted with LiFi



(c) Time to solve

FIGURE 15. Quality metrics for all proposed algorithms in the Small scenario with URLLC, Factory, and AI applications.

Blackbox optimization techniques are very similar, which reinforces the claim that treating the objective function as a blackbox yields similar results, which are no better than a greedy approach. The Expert offloads URLLC tasks mostly through WiFi since the offloading destination selected is mostly the WiFi AP if not processed locally. The AI tasks, on the other hand, are offloaded further away to the Router or Cloud due to their higher processing capacity and utilize more of the LiFi link due to its higher data rate.

The tasks of Factory application are processed completely locally. Similarly, the DRL-based approach processes the URLLC and Factory tasks locally, while the AI application is offloaded across multiple types of destinations. However, contrary to the Expert, it does offload a majority of the AI tasks through WiFi and this is reflected in the Latency QoS for the AI application.

While the top three algorithms in terms of network performance have been identified previously, it is also necessary to compare them in terms of the time taken to solve the optimization problem. Fig. 15(c) visualizes this solve time metric in seconds and the time comparison is shown in the logarithmic scale due to the wide range of times. The Expert stands out with its longest solve time, making it infeasible for a practical implementation. A similar conclusion can be made about the Extra Trees and Random Forest. The meta-heuristic, as already shown in Fig. 8(b) and Fig. 9(b), offers a solve time of a few seconds which is much better but is still inefficient for a real-world implementation. While the Greedy algorithm offers a lower solve time, its network performance is poor, as already seen in Fig. 15(a). The DRL solution turns out to be the most efficient in terms of prediction time, which is significantly faster than the other approaches. This is evidenced by the results of the statistical tests annotated on the figure comparing the DRL approach with all others. This speed makes the DRL solution especially attractive as the best compromise between network performance and solve time, making it an optimal choice for real-time implementation on actual hardware.

The energy consumption of an algorithm is another aspect to consider before deciding on the algorithm to implement. We exclude the MINLP solver-based algorithm and the blackbox optimization techniques from this comparison since the solver-based solution is too complex for a larger scenario and the blackbox techniques have poor network performance. Comparing the GA and DRL solutions, the GA takes 4.244×10^{-6} kWh on average for one run of the optimization and the DRL 0.0674 kWh. The DRL has a one-time overhead of the training and cross-validation, which consumes most of the energy. If this can be implemented on a more powerful, less energy-constrained machine, then the model that is installed during the run time of the algorithm (on the testing data) only consumes 3.989×10^{-7} kWh of energy for one run of the optimization. The energy is calculated using the Running Average Power Limit interface on Intel processors and NVIDIA-smi for the GPU.

VII. CONCLUSION

This paper presented ComputiFi, a framework designed to enhance task offloading efficiency in LiFi-WiFi networks, tailored specifically for reducing latency in latency-critical applications. Through a combination of advanced optimization tools, a dynamic resource allocation strategy across varied computational servers, and multipath transmissions, ComputiFi addressed the challenges posed by heterogeneous network environments. Additionally, we also presented a multihop latency model that is suitable for data flows with multiple packets. The results demonstrate that our approach not only reduces latency but also maintains high QoS across various network architectures, confirming its practical viability. Our proposed approach to task offloading achieves 69.3% lower user average latency compared to the best-performing baseline approach in a Small scenario with eMBB, Factory, and AI applications and 65.8% lower user average latency with URLLC, Factory, and AI applications.

The comprehensive evaluation of various optimization algorithms for task offloading in multipath, multihop, LiFi-WiFi networks highlights the strengths and limitations of each approach. The GA and DRL-based solutions emerge as top performers, with the DRL solution performing the best in speed with prediction times in the order of a few milliseconds, making it ideal for real-time applications while still achieving 40.23% lower user average latency than the best baseline approach with URLLC, Factory, and AI applications. These findings emphasize the importance of selecting the right optimization strategy based on specific network and operational requirements.

VIII. FUTURE WORK

This paper establishes a strong foundation for optimizing task offloading in multipath, multihop LiFi-WiFi networks, opening several avenues for further research. Integrating more sophisticated machine learning techniques, particularly adaptive learning models that evolve based on real-time network data, could dynamically adjust to changes in user behavior, network congestion, and other environmental factors. An important area for future exploration involves incorporating energy efficiency metrics into the optimization problem to balance latency and energy consumption, which is particularly beneficial for battery-dependent mobile devices and IoT applications. Enhancing the framework to more effectively manage bandwidth allocation can ensure optimal use of available network resources, addressing both throughput needs and congestion management. Additionally, conducting studies on real-world implementations could provide insights into the practical challenges and performance under actual operating conditions.

REFERENCES

[1] "5G; service requirements for cyber-physical control applications in vertical domains," 3GPP, Sophia Antipolis, France, Rep. 22.104, May 2022.
[2] "5G; study on scenarios and requirements for next generation access technologies," 3GPP, Sophia Antipolis, France, Rep. 38.913, May 2022.
[3] M. Ayyash et al., "Coexistence of WiFi and LiFi toward 5G: Concepts, opportunities, and challenges," *IEEE Commun. Mag.*, vol. 54, no. 2, pp. 64–71, Feb. 2016.
[4] H. Haas, L. Yin, Y. Wang, and C. Chen, "What is LiFi?" *J. Lightw. Technol.*, vol. 34, no. 6, pp. 1533–1544, Mar. 15, 2016.
[5] X. Wu, M. D. Soltani, L. Zhou, M. Safari, and H. Haas, "Hybrid LiFi and WiFi networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1398–1420, 2nd Quart., 2021.
[6] W. Fan et al., "Joint task offloading and service caching for multi-access edge computing in WiFi-cellular heterogeneous networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 11, pp. 9653–9667, Nov. 2022.

[7] X. Xu and Y. Song, "A deep reinforcement learning-based optimal computation offloading scheme for VR video transmission in mobile edge networks," *IEEE Access*, vol. 11, pp. 122772–122781, 2023.
[8] J. Yun, Y. Goh, W. Yoo, and J.-M. Chung, "5G multi-RAT URLLC and eMBB dynamic task offloading with MEC resource allocation using distributed deep reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 20, pp. 20733–20749, Oct. 2022.
[9] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883–893, Jun. 2020.
[10] A. Suzuki, M. Kobayashi, and E. Oki, "Multi-agent deep reinforcement learning for cooperative computing offloading and route optimization in multi cloud-edge networks," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 4, pp. 4416–4434, Dec. 2023.
[11] J. von Mankowski, E. Durmaz, A. Papa, H. Vijayaraghavan, and W. Kellerer, "Aerial-aided multiaccess edge computing: Dynamic and joint optimization of task and service placement and routing in multilayer networks," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 59, no. 3, pp. 2593–2607, Dec. 2023.
[12] G. Feng, X. Li, Z. Gao, C. Wang, H. Lv, and Q. Zhao, "Multi-path and multi-hop task offloading in mobile Ad Hoc networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 6, pp. 5347–5361, Jun. 2021.
[13] Y. Deng, H. Zhang, X. Chen, and Y. Fang, "Multi-hop task routing in vehicle-assisted collaborative edge computing," *IEEE Trans. Veh. Technol.*, vol. 73, no. 2, pp. 2444–2455, Feb. 2024.
[14] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, and L. Guo, "Computation offloading in mobile edge computing networks: A survey," *J. Netw. Comput. Appl.*, vol. 202, Jun. 2022, Art. no. 103366.
[15] U. Saleem, Y. Liu, S. Jangsher, Y. Li, and T. Jiang, "Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 360–374, Jan. 2021.
[16] Y. Liu, Y. Mao, Z. Liu, and Y. Yang, "Deep learning-assisted online task offloading for latency minimization in heterogeneous mobile edge," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 4062–4075, May 2024.
[17] X. Chen and G. Liu, "Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10843–10856, Jul. 2021.
[18] T. Braud, P. Zhou, J. Kangasharju, and P. Hui, "Multipath computation offloading for mobile augmented reality," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. (PerCom)*, 2020, pp. 1–10.
[19] H. Zhang, J. Guo, L. Yang, X. Li, and H. Ji, "Computation offloading considering fronthaul and backhaul in small-cell networks integrated with MEC," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2017, pp. 115–120.
[20] A. Jano, M. Mert Bese, N. Mohan, W. Kellerer, and J. Ott, "nextGSIM: Towards simulating network resource management for beyond 5G networks," in *Proc. IEEE Future Netw. World Forum*, 2023, pp. 1–7.
[21] Y. Liu, Y. Mao, Z. Liu, F. Ye, and Y. Yang, "Joint task offloading and resource allocation in heterogeneous edge environments," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 7318–7334, Jun. 2024.
[22] Y. Wang, X. Wu, and H. Haas, "Load balancing game with shadowing effect for indoor hybrid LiFi/RF networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 4, pp. 2366–2378, Apr. 2017.
[23] X. Wu and H. Haas, "Load balancing for hybrid LiFi and WiFi networks: To tackle user mobility and light-path blockage," *IEEE Trans. Commun.*, vol. 68, no. 3, pp. 1675–1683, Mar. 2020.
[24] F. Firyaguna, J. Kibilda, C. Galiotto, and N. Marchetti, "Performance analysis of indoor mmWave networks with ceiling-mounted access points," *IEEE Trans. Mobile Comput.*, vol. 20, no. 5, pp. 1940–1950, May 2021.
[25] M. D. Soltani, A. A. Purwita, Z. Zeng, H. Haas, and M. Safari, "Modeling the random orientation of mobile devices: Measurement, analysis and LiFi use case," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 2157–2172, Mar. 2019.
[26] (Gurobi Optim., LLC, Beaverton, OR, USA). *Gurobi Optimizer Reference Manual*. (2024). [Online]. Available: https://www.gurobi.com
[27] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley, 1989.
[28] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization* (Natural Computing Series). Berlin, Germany: Springer, 2005.
[29] R. Hooke and T. A. Jeeves, "'Direct search" solution of numerical and statistical problems," *J. ACM*, vol. 8, no. 2, pp. 212–229, 1961.
[30] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. ICNN*, 1995, pp. 1942–1948.
[31] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, pp. 284–294, Sep. 2000.
[32] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.
[33] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, pp. 3–42, Mar. 2006.
[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[35] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html
[36] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation Hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2019, pp. 2623–2631.
[37] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 1947. [Online]. Available: https://doi.org/10.1214/aoms/1177730491
[38] F. Charlier et al. "Statannotations." Oct. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7213391
[39] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *J. Roy. Statist. Soc., Ser. B*, vol. 57, no. 1, pp. 289–300, 1995.

HANSINI VIJAYARAGHAVAN (Graduate Student Member, IEEE) received the M.Sc. degree in communications engineering from the Technical University of Munich, Munich, Germany, in 2018, where she is currently pursuing the Ph.D. degree in communications engineering with the Chair of Communication Networks.

Her research interests include the design and management of LiFi–WiFi heterogeneous networks, specifically environment planning and wireless resource allocation.

JÖRG VON MANKOWSKI (Graduate Student Member, IEEE) received the B.Eng. degree in electrical engineering and information technology and the M.Sc. degree in electrical engineering from the Munich University of Applied Sciences, Munich, Germany, in 2011 and 2014, respectively. He is currently pursuing the Ph.D. degree in resource allocation for aircraft-related communication with the Chair of Communication Networks, Technical University of Munich, Munich.

His research interests include the design and analysis of aeronautical communication, channel modeling, and interference management in multitechnology networks.

WOLFGANG KELLERER (Senior Member, IEEE) received the Dipl.Ing. (master's) and Dr.Ing. (Ph.D.) degrees in electrical and computer engineering from the Technical University of Munich (TUM) in 1995 and 2002, respectively.

He is a Full Professor with TUM, heading the Chair of Communication Networks with the Department of Electrical and Computer Engineering. Before, he was with NTT DOCOMO's European Research Laboratories for over ten years.

Prof. Kellerer currently serves as an Associate Editor for IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and the Area Editor for Network Virtualization for IEEE COMMUNICATIONS SURVEYS AND TUTORIALS.