SPARQ: An Optimization Framework for the Distribution of AI-Intensive Applications under Non-Linear Delay Constraints

Pietro Spadaccino, Paolo Di Lorenzo, Sergio Barbarossa, Antonia M. Tulino, Jaime Llorca

Abstract—Next-generation real-time compute-intensive applications, such as extended reality, multi-user gaming, and autonomous transportation, are increasingly composed of heterogeneous AI-intensive functions with diverse resource requirements and stringent latency constraints. While recent advances have enabled very efficient algorithms for joint service placement, routing, and resource allocation for increasingly complex applications, current models fail to capture the non-linear relationship between delay and resource usage that becomes especially relevant in AI-intensive workloads. In this paper, we extend the *cloud network flow* optimization framework to support queuing-delay-aware orchestration of distributed AI applications over edge-cloud infrastructures. We introduce two execution models—Guaranteed-Resource (GR) and Shared-Resource (SR)—that more accurately capture how computation and communication delays emerge from system-level resource constraints. These models incorporate M/M/1 and M/G/1 queue dynamics to represent dedicated and shared resource usage, respectively. The resulting optimization problem is non-convex due to the non-linear delay terms. To overcome this, we develop SPARQ, an iterative approximation algorithm that decomposes the problem into two convex subproblems, enabling joint optimization of service placement, routing, and resource allocation under nonlinear delay constraints. Simulation results demonstrate that the SPARQ not only offers a more faithful representation of system delays, but also substantially improves resource efficiency and the overall cost-delay tradeoff compared to existing state-of-the-art methods.

Index Terms—Edge computing, service function chain, service graph, service placement, resource allocation, cloud network flow
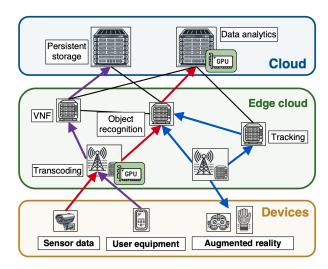
Fig. 1: Edge-cloud architecture. Various service chains running over a shared distributed computing network. Network elements in the edge-cloud architecture may be equipped with CPU and/or GPU processing capabilities, to accommodate AI workloads.

I. INTRODUCTION

The proliferation of distributed cloud architectures, such as Multi-access Edge Computing (MEC) and Fog Computing, has enabled innovative service models capable of addressing the stringent latency and computational demands of AI and next-generation applications [1], [2]. These applications, including augmented reality, autonomous driving, and industrial automation, require the efficient coordination of communication, computation, and storage resources. Service Function Chains (SFCs) or, more generally, service graphs, which define sequences of interdependent service functions, are crucial for meeting these demands. Properly embedding these SFCs across distributed cloud infrastructures is vital for optimizing performance while minimizing resource consumption. Existing research has largely focused on static and dynamic placement and optimization of SFCs, treating the orchestration of service chains as a combinatorial optimization problem aimed at minimizing costs or maximizing throughput. Due to the complexity and non-linearity of the system, however, approximations have been used which, depending on the case, can be too rough or stringent. Moreover, SFC optimization must consider the growing data requirements of these applications, along with the inherent non-linearity of network delays under stochastic traffic conditions.

To address the complexities and requirements of SFCs, edge-cloud architectures are introduced. Fig. 1 illustrates an example of this architecture, where services from multiple SFCs are distributed across various nodes in the network. In our modeling framework, an SFC is represented by a Directed Acyclic Graph (DAG), whose vertices represent service functions or tasks, and edges represent *commodities* or data streams flowing through the network.

Numerous applications can benefit from such an architecture, including tasks like image and text recognition, data analytics, and video stream processing. Decisions around

function placement, flow routing, and resource allocation must carefully consider the cost of operating the required communication and computations resources, as well as the delay incurred in transmitting data along communication links, and in processing data by the service functions hosted at the edge-cloud computation resources.

A. Motivation

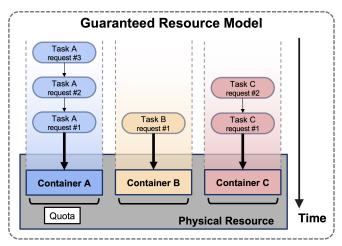When designing the placement, routing, and resource allocation of distributed applications, it is also crucial to consider the underlying computing architecture and deployment constraints. This is particularly relevant for AI-based applications, which impose diverse and stringent computational requirements. AI workloads typically involve a combination of CPU-intensive tasks—such as data pre-processing, feature extraction, and general computations—and GPU-intensive tasks, which primarily involve executing deep learning models for inference or training. The latter often require substantial VRAM capacity, as modern AI models tend to be large [3], fully occupying the available GPU memory for the duration of execution. When an AI model is loaded onto a GPU-equipped node, it fully occupies the GPU memory, preventing concurrent execution of other models on the same device until processing is complete. Conversely, CPU-only tasks may either serve as AI pre-processing steps or represent general-purpose computations unrelated to AI, such as database queries, logging, or application control logic. Such operations typically fit comfortably within the system RAM, enabling multiple processes to reside simultaneously in memory. The operating system efficiently manages these tasks by rapidly performing context-switching, which involves temporarily pausing one task and resuming another. Because these context switches occur at very high frequencies, the user perceives that multiple CPU tasks are executed concurrently, despite the CPU physically processing instructions sequentially.

Expanding the discussion beyond just CPU and GPU, we can identify two general classes of computational workloads, schematized in Fig. 2. The first class, as reported in Fig. 2a, comprises tasks that fully utilize the available computational resources, effectively blocking or monopolizing them during execution. Such tasks prevent simultaneous use of these resources by other tasks, leading to a scenario where concurrent execution or interleaving of multiple tasks is not possible. In this case, the same physical resource is *shared* across several tasks. Examples include tasks running large AI models fully occupying a GPU's VRAM, or specialized computations on dedicated hardware like FPGA accelerators. The second class includes tasks that only partially utilize the available resources, as reported in Fig. 2b, allowing multiple such tasks to coexist simultaneously without resource contention. Due to their limited resource requirements, these tasks can efficiently run alongside others within the same computational environment, benefiting from the operating system's task scheduling and multiplexing capabilities. Examples include lightweight CPU-bound processes, lightweight GPU processing, containerized microservices, or any task executed within *guaranteed* resource quotas that ensure minimal interference with others.

(a) Heterogeneous tasks access the same physical resource (e.g., a GPU) without resource-level isolation. Different tasks compete for the shared resource, leading to interdependent delays.

(b) Heterogeneous tasks are allocated a fixed quota of the physical resource (e.g., CPU time slice or reserved GPU memory), enabling isolated execution. Tasks are processed independently within their assigned quota, leading to non-interdependent delays.

Fig. 2: Model of a shared resource over time (a) and a resource with guaranteed quotas over time (b).

B. Related work

The optimization of SFCs in distributed cloud networks has been extensively studied in recent years, particularly in the context of minimizing resource operational cost while meeting quality of service (QoS) requirements (e.g., latency). Several models have been proposed to capture the flow of requests and the placement of services within distributed computing networks. Among these, the Cloud Network Flow (CNF) model introduced in [4]–[6] stands out. This model represents Service Function Chains (SFCs) as Directed Acyclic Graphs (DAGs), allowing for an efficient representation of complex service workflows. Distributed computing networks are then modeled via *cloud-augmented graphs*, where in addition to communication links representing data transmission capabilities, there are production, consumption, and computation links representing the capability to produce, consume, and process data throughout he network. By leveraging this structure, the model enables the joint optimization of placement, routing, and resource allocation by solving a single flow problem on the cloud-augmented graph. The CNF model has been widely adopted for the study of distributed cloud-edge networks with embedded service DAGs, accommodating both static

optimization and dynamic control scenarios. Static service optimization, including placement, routing, and resource allocation, has been explored in works such as [4]–[9], addressing varying levels of service complexity and resource demands. However, these studies do not account for queueing delays. In contrast, studies like [10], [11] focus on dynamic service control policies that optimize routing, processing, and scheduling decisions based on local real-time observations. While these approaches integrate queueing systems and account for associated delays, they primarily aim to achieve bounded average delay through queue stability, without explicitly guaranteeing end-to-end service latency.

Various studies have proposed models to address the challenge of mapping service chains onto network resources, often leveraging queuing theory to capture traffic dynamics. For instance, [12] models communication links as queues and optimizes queue throughput by stabilizing the Lyapunov drift function. While this approach effectively addresses resource allocation, it does not consider the placement or routing of SFCs. In [13], the authors explore dynamic service control in edge networks, addressing both live and static data streams. They employ a graph-based model to represent network processing and transmission resources, with queues managing Poisson-distributed arrival rates. This approach facilitates the handling of dynamic traffic flows and ensures throughput-optimal performance. In [14], the authors develop a fully distributed packet processing and routing policy for multicast services. Traffic across multiple network links is modeled using queues, enabling dynamic control of packet processing and duplication. This method captures the behavior of data flows traversing the network, ensuring throughput-optimal routing and efficient resource utilization. Also, [15] introduces a dynamic computation offloading strategy tailored for MEC environments, with a focus on ultra-reliable low-latency communications (URLLC). The approach utilizes queuing models to dynamically manage communication and computation tasks, optimizing the joint allocation of radio and computational resources. By ensuring that the sum of communication and computation queue lengths remains within a probabilistic bound, their algorithm balances service delay and energy consumption while adhering to reliability constraints. This stochastic optimization approach effectively addresses latency requirements under dynamic network conditions.

In addition to focusing solely on modeling and embeddings, several works in the literature also take into account practical and implementation-related challenges. These studies address, among other things, the issue of memory limitations in network routers, specifically Ternary Content Addressable Memories (TCAMs) [16], [17], which are crucial for performing SFC placement and request classification in NFV/SDN environments. Authors in [18], by proposing an offloading strategy, use an Integer Linear Programming (ILP) formulation and heuristic approaches to efficiently handle SFC requests under TCAM size constraints. This approach is more practical as it directly addresses the hardware limitations that impact SFC deployment, offering a scalable solution to increase the number of classified requests without significantly affecting network Quality of Service (QoS). Moreover, in [19], the authors

address the critical challenge of efficiently placing Virtual Network Functions (VNFs) SFCs and allocating resources in 5G networks, specifically tailored to vertical industries. They propose a queuing-based model to map service requirements into decisions about VNF placement, CPU assignment, and traffic routing. Their strategy is particularly suited for practical implementations in 5G networks, where it dynamically adjusts resource allocation based on traffic loads, ensuring low latency and efficient use of network resources.

C. Main contributions

In this paper, we propose **SPARQ** (Service Placement, Resource Allocation and Routing with Queue-Aware Delays), a **comprehensive optimization framework to jointly optimize service function placement, flow routing, and computation-communication resource allocation** with the goal of minimizing overall operational cost while guaranteeing resource capacity and end-to-end latency constraints. Importantly, **latency constraints capture the critical nonlinearities associated with the delay experienced by next-generation AI-intensive applications** consuming heterogeneous shared and guaranteed resources.

In particular, we propose **two resource models, named Guaranteed-Resource (GR) and Shared-Resource (SR) models**, which offer a more accurate representation of how tasks are executed on modern hardware. The GR model ensures that tasks are allocated dedicated resources, preventing contention and allowing each task to be processed independently. In contrast, the SR model models scenarios where tasks must share a resource, and the delays experienced by each task depend on the usage of the resource by other tasks. This distinction is crucial to accurately model real-world scenarios, especially in the context of emerging applications with an increasing presence of AI workloads, where resource contention (e.g., GPU memory) plays a significant role in overall system performance.

To incorporate these models into the CNF framework, we propose a queue-based modeling approach for the computation and communication links within the cloud-augmented graph. This approach models **each communication and computation system** (links in the cloud-augmented graph) **as a mixture of M/G/1 and M/M/1 queues** to accurately capture the delay experienced by service commodities (or data streams) flowing through the network. Our model precisely expresses delay as a function of service placement, incoming request volume, and the resources available to serve these requests. Unlike the private delay model[1] assumed in previous approaches, this flexible queue-based approach more accurately captures service delays in stochastic AI-intensive applications. Moreover, prior works use queuing models primarily for communication links. In our approach, leveraging the cloud-augmented graph, **we extend queue delay modeling to all network components** (communication and computation

---

[1]In the private delay, model, delay is assumed constant with respect to allocated resources, as long as allocated resources are above average load, hence ignoring the queuing and congestion effects due to stochastic traffic variations.
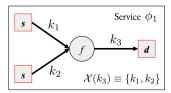
Fig. 3: Example of a service graph. Commodity k_3 is produced requiring as input commodities k_1, k_2.

resources) creating a more comprehensive and realistic representation of the compute-communication system.

However, direct optimization using this model is challenging due to its non-convex formulation. To address this, we develop several **approximation and convexification techniques** specific to our model. These methods are **integrated into a single iterative algorithm that jointly optimizes service placement, routing, and resource allocation with accurate end-to-end latency constraints**.

To assess the performance of the proposed framework, we perform extensive simulations on representative edge-cloud deployment scenarios and realistic traffic patterns. Simulation results demonstrate that the approximated delay model closely mirrors the *a-posteriori* measured delay, validating the accuracy of the approximation and confirming its suitability for solving the optimization problem without significant loss in solution fidelity. Moreover, the proposed approach consistently outperforms baseline methods in both cost efficiency and delay compliance, yielding a substantially improved cost-delay tradeoff. Taken together, these results confirm the practical viability of our method and clearly establish its advantage over existing solutions for managing complex, latency-sensitive AI workloads.

II. SYSTEM MODEL

In this section, we formalize the network and service models that underpin the Cloud Network Flow (CNF) framework for optimizing the deployment of distributed services in edge-cloud environments.

At the core of this framework lies a *multi-commodity-chain flow* problem [20], where each commodity k represents a data flow associated with a specific service component. In the classical multi-commodity-flow formulation, a commodity is defined by a fixed source-destination pair (s, d), corresponding to a flow of data between two network nodes.

In the CNF setting, however, this notion is extended to capture the flexibility required by modern service function chains and AI workflows. In particular, some commodities correspond to intermediate service stages whose execution locations are not predefined. For these flows, their source and/or destination nodes are treated as decision variables and are resolved by the optimization process, which jointly determines the placement of service components and the routing of data between them. In this context, we say that commodity k is *produced* at a node u either because u is the origin of k (e.g., a data source such as a sensor), or because k is produced as the result of a computation at u involving one or more input commodities. Similarly, we say k is *consumed* at a node v either because v

is the final destination of the data (e.g., a user device or data sink), or because k is used as input to a computation at node v that generates new downstream commodities.

Each commodity represents a distinct data stream (or data flow) and is associated with possibly heterogeneous communication and computation requirements (e.g., bandwidth, latency sensitivity, processing load), thereby reflecting the diverse service-level demands of modern applications.

To effectively capture the interplay between the physical infrastructure, the service logic, and the associated data flows, the CNF framework introduces three distinct graph abstractions, each serving a complementary role in the overall modeling approach: i) the *network graph*, which represents the physical infrastructure of the network; ii) the *service graph*, representing the computations required by the application; iii) and the *cloud-augmented* graph, which extends the network graph with additional auxiliary nodes added to represent computation, production and consumption operations. The network graph is modeled as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ represent the set of network nodes and links, respectively. For any two nodes $u, v \in \mathcal{V}$, an link $(u, v) \in \mathcal{E}$ indicates a network connection.

As for the service graph, we define $\mathcal{K}$ to be the set of all commodities. A service $\phi \in \Phi$ is modeled as a DAG, where edges correspond to the commodities required by the service and vertices represent processing steps, referred to as *service functions*. Specifically, each commodity $k \in \phi$ represents a data flow between two service functions and is therefore associated with a directed edge in the DAG. This edge indicates that the output of one function serves as the input to the next. Considering commodities $k, j \in \mathcal{K}$, if commodity j is required to produce commodity k, this dependency is expressed as $j \in \mathcal{X}(k)$. We call k a source commodity if it is generated with no inputs $\mathcal{X}(k) = \emptyset$ in a network node $s(k) \in V$, and denote by $\mathcal{K}^s \in \mathcal{K}$ the set of source commodities. Similarly, we call k a destination commodity if it is consumed without generating any output at node $d(k) \in V$, and denote by $\mathcal{K}^d \in \mathcal{K}$ the set of destination commodities.

Embedding a service ϕ in the network graph means mapping each service function to a physical node (*function placement*) and each commodity to a feasible path between them (*data routing*). Moreover, we note that multiple services $\phi_1, \phi_2, \ldots \in \Phi$ may require to be embedded within the same network graph. We use $\phi(k)$ to denote the service associated with commodity k. An example of a simple service graph is reported in Fig. 3.

To perform an efficient embedding, we make use of the cloud-augmented graph model [4]–[6]. The augmented graph of $\mathcal{G}$ is represented by $\mathcal{G}^a = (\mathcal{V}^a, \mathcal{E}^a)$. It is constructed by creating additional nodes for each communication node $u \in \mathcal{V}$. These nodes are denoted by p, s, and d, modeling the computation, production, and consumption capabilities of node u, respectively, as illustrated in Fig. 4. In general, a node $u \in \mathcal{V}$ may have more than one computation node: in our model, the computation can be carried out by different types of computational systems $\{p_1, p_2, \ldots, p_N\}$. These computational systems could, for example, be a specific type of Amazon Web Services (AWS) EC2 machine (e.g., small or xl), or a
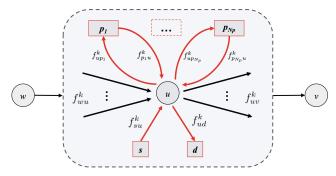
Fig. 4: Augmented graph node. For each node $u \in \mathcal{V}$, virtual nodes are created, representing production, consumption, and computation capabilities. The augmentation enables the formulation of the service graph embedding problem within the augmented graph as a single network flow problem.

machine equipped with additional hardware such as a GPUs or FPGAs. A computational system p_i can provide one or more physical resources in $\mathcal{R} = \{r_1, r_2, \dots\}$, representing, for example, CPU, GPU or FPGA capabilities. Node u can offer zero or more computational systems $\{p_i, p_j, \dots\}$, with computation links (u, p_i) and $(u, p_i) \in \mathcal{E}^a$ existing if node u can offer system p_i.

Depending on the roles of nodes u and v, an edge $(u, v) \in \mathcal{E}^a$ may represent a communication link, a computation link, or a data transfer to or from storage. This augmentation of the network graph enables the formulation of the service graph embedding problem as a *network flow problem* [4], [21], making it suitable for tractable optimization through established flow-based techniques.

We define binary **flow variables** $f_{uv}^k \in \{0, 1\}$ to indicate whether commodity $k \in \mathcal{K}$ is assigned to link $(u, v) \in \mathcal{E}^a$. These variables jointly encode both **function placement** and **data routing** over the cloud-augmented graph. Concretely, if u is a computation node, then $f_{uv}^k = 1$ models the placement of the function that produces commodity k at the computation node u. If both u and v are communication nodes, then $f_{uv}^k = 1$ denotes that commodity k is routed along the communication link (u, v).

The other key decision variables in the addressed service optimization problem are the **resource allocation variables**, which are introduced in Section III.

A. Computational Architecture for AI Workloads

In AI-focused server environments, computation primarily occurs within containers and virtual machines running on dedicated hardware. Containers are favored because they offer strong isolation, near-native performance, and fine-grained resource control. A single server $u \in \mathcal{V}$ may host multiple containers, each serving a different application component. Whenever two commodities $k, l \in \mathcal{K}$ satisfy $l \in \mathcal{X}(k)$, we interpret l as an input commodity that, through a service function executed in a container, contributes to the production of commodity k. In our model, each container processes exactly one commodity at a time, while the underlying operating system multiplexes containers via time-sharing.

Understanding how computational tasks are executed within containers is crucial for accurately modeling delays, as these delays vary significantly depending on the type of workload. In particular, it is essential to distinguish between CPU-bound and GPU-bound computations, which exhibit fundamentally different execution and scheduling behaviors.

CPU-based workloads can support multiple concurrent tasks in memory by leveraging fast context switching, which enables efficient time-sharing. This capability allows the operating system to define and guarantee processing quotas for each commodity, ensuring fair resource allocation and predictable performance. As a result, queuing delays in CPU-bound scenarios are primarily a consequence of time-sharing.

Conversely, GPU computation, especially in AI applications, operates under a different paradigm. Specifcially, modern AI workloads executed on GPUs load large deep-learning models that often saturate VRAM [22], [23]. Unlike CPU scheduling, switching between different AI models would require offloading and reloading model data into VRAM, introducing substantial I/O overhead and inefficiencies. As a result, a GPU effectively runs one model at a time, making its delay dynamics very different from those of a CPU.
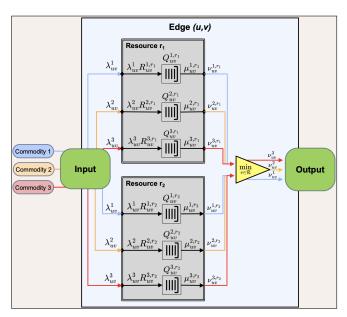
Abstracting beyond the CPU/GPU dichotomy, we identify two computation-delay regimes: i) guaranteed-quota regime where every task receives a dedicated slice of the resource, so its sojourn time is independent of other tasks on the same node; ii) shared-resource regime where multiple tasks contend for a common resource pool, and each task's delay depends on the aggregate load.

These regimes necessitate two distinct queuing models. Accordingly, we choose the appropriate queue class based on the architecture and physical characteristics of the node or link (u, v) to accurately reflect the corresponding delay behavior. The formal delay expressions for both regimes are provided in the next section.
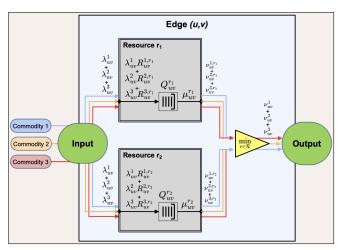
III. QUEUE-BASED DELAY MODELING

In this section, we aim to develop a model for estimating the expected latency experienced by a commodity as it traverses the network. This latency comprises the waiting and processing delays encountered by requests across the network. To model these delays, we represent each link $(u, v) \in \mathcal{E}^a$ with one or more queues Q_{uv}. In the following, we assume that each queue possesses a single server, is non-preemptive, and is work-conserving, with a First-Come First-Served (FCFS) scheduling policy.

We begin by characterizing the total average rate that may traverse link (u, v), followed by two alternative models for the service rate, and finally derive the expected delay experienced by requests crossing (u, v). Specifically, we assume that requests for a global service ϕ arrive according to a Poisson process with rate Λ^ϕ. Each service ϕ is modeled as a directed acyclic graph (DAG) of commodities. The request arrival rate for all commodities $k \in \phi$ is given by the service request rate Λ^ϕ. The **request arrival rate** of commodity k at link (u, v) is hence given by $\lambda_{uv}^k = f_{uv}^k \Lambda^{\phi(k)}$, where recall that $\phi(k)$ denotes the service associated with commodity k.

(a) Guaranteed-Resource model. Every incoming request has its own commodity-dedicated queue inside the system. Each queue system serves only one commodity. The system time experienced by one commodity does not influence other commodities.

(b) Shared-Resource model. All requests flow inside the same queue, shared across commodities. The service requirements vary by commodity, resulting in a shared processing resource where delays depend on the combined workload. The system time experienced by one commodity is influenced and depends from other commodity using the same shared resource.

Fig. 5: Flow rates for the Shared and Guaranteed Resource models.

Accordingly, the total request arrival rate on link (u, v) is given by $\lambda_{uv} = \sum_{k \in \mathcal{K}} \lambda_{uv}^k$.

In the following subsections, we present two strategies for modeling **resource allocation and associated delays**, illustrated in Fig. 5, and provide a detailed description of each.

A. Guaranteed-Resource model

In this first approach, we assume that resources are exclusively allocated to each container processing a commodity, provided that the total allocated resources remain within the available capacity, as illustrated in Fig. 5a. This is referred to as the *Guaranteed-Resource* (GR) model. Under this model, multiple processing containers can coexist on the same server without interference or contention, since each is assigned a dedicated portion of the available resources. This setup enables us to model processing using distinct and independent queues, each dedicated to a single commodity. This model is particularly suitable for cases such as:

- CPU processing within containerized environments, where each container is allocated a guaranteed CPU time slice, ensuring that its processing delays are independent of other containers;
- Network slices on communication infrastructure, where pre-provisioned network slices with fixed bandwidth allocations, where resources are reserved and isolated for specific services or tenants;
- Dedicated virtual machines or dedicated bare-metal servers, where computational resources are fully assigned and not shared dynamically with other workloads.

The set of links following the GR model is indicated as $\mathcal{E}^{a,\mathrm{GR}} \subseteq \mathcal{E}^a$. We define the decision variables that govern the **resource service rate** at which $(u, v) \in \mathcal{E}^{a,\mathrm{GR}}$ serves commodity k using resource r as $\mu_{uv}^{k,r}$. Depending on the nature of resource r, it may be quantified in e.g., bits per second, operations per second, CPU/GPU cycles per second, etc.

The demand for resource $r \in \mathcal{R}$ by commodity k on link (u, v) is denoted by $R_{uv}^{k,r}$ and is referred to as the **resource-r requirement** (or workload) of k on (u, v). Depending on the nature of resource r, it may be quantified in units of e.g., bits per request, operations per request, CPU/GPU cycles per request, etc.

We can then derive the **request service rate** of commodity k on (u, v), i.e., the rate at which (u, v) serves requests of commodity k, as $\nu_{uv}^k = \min_{r \in \mathcal{R}} \frac{\mu_{uv}^{k,r}}{R_{uv}^{k,r}}$, measured in requests/s. The minimization over r is needed because commodity k may require service from multiple resources, and the overall output rate is constrained by the slowest rate among all resources, as shown in Fig. 5a. The service time for requests of commodity k on link (u, v) is modeled as a negative exponential distribution with rate ν_{uv}^k. Finally, given that the arrival process for requests of commodity k on (u, v) follows a Poisson distribution with parameter λ_{uv}^k. we can model link (u, v) as a set of **M/M/1** queues $\{Q_{uv}^{k,r}\}$, with a separate queue for each commodity-resource pair (k, r).

B. Shared-Resource model

In contrast to the GR model, the second approach assumes that computation relies on a single shared resource that is concurrently used by multiple commodities, as illustrated in Fig. 5b. This is referred to as the *Shared-Resource* (SR) model. In this setting, resources are not partitioned among containers; instead, all active commodities compete for the same processing resources. As a result, the sojourn time of any given commodity is affected by the collective load imposed by all other commodities sharing the same resource. This model is particularly suitable for environments such as:

- GPU processing for heavy AI workloads, where multiple large AI models share the same GPU, which cannot efficiently interleave executions due to VRAM constraints, causing task delays to depend on all workloads on the GPU;
- Communication links without network slicing, where all traffic shares the same physical medium, leading to contention and interdependent delays among commodities routed on the link;
- Shared storage or I/O resources, where multiple processes access the same disk or network storage, leading to queuing delays affected by concurrent usage.

The set of (u, v) following the SR model is indicated as $\mathcal{E}^{a,\text{SR}} \subseteq \mathcal{E}^a$. In this case, we define the decisions variables that govern the **resource service rate** at which $(u, v) \in \mathcal{E}^{a,\text{SR}}$ serves *all* arriving commodities over resource r as μ_{uv}^r. The arrival process of the SR model is the cumulative process obtained by summing the arrivals for all commodities routed on (u, v), which, like in the GR model, follows a Poisson distribution. The service time of requests for commodity k is modeled as a negative exponential distribution with rate $\nu_{uv}^k = \min_{r \in \mathcal{R}} \frac{\mu_{uv}^r}{R_{uv}^{k,r}}$, named request service rate. In contrast to the GR model, different commodities $j \neq k$ can be processed by the same queue. Since each commodity k, j traversing (u, v) requires different workloads $R_{uv}^{k,r}, R_{uv}^{j,r}$, the cumulative service time is general and doesn't follow an exponential distribution. Hence, given that the service request arrival process is Poisson and the individual commodity service times are exponential, a given network queue Q_{uv}^r can be well-approximated by an **M/G/1** model for large networks [20]. Therefore, a link (u, v) is modeled with a set of M/G/1 queues $\{Q_{uv}^r\}$, one for each resource, but shared among commodities.

A summary of the parameters and variables introduced in this section is given in Table I.

C. Delay modeling

We now analyze the delay associated with a generic queue in the system. Let $D_{uv}^{k,r}$ be a random variable expressing the sojourn time (queue time plus processing time) of commodity $k \in \mathcal{K}$ requiring resource-r traversing link $(u, v) \in \mathcal{E}^a$. Following the proposed models, we investigate the sojourn time $D_{uv}^{k,r}$ in four distinct cases, distinguishing whether (u, v) is a communication or computation link, and whether it follows the GR or SR model. We begin by analyzing communication links the GR and SR models. Then, we generalize the obtained formulation to analyze the computation links within the GR and SR models.

1) Communication links, GR model. Let $(u, v) \in \mathcal{E}^a$ denote a communication link modeled under the GR framework. We model communication links with a single resource type $r \in \mathcal{R}$, which could be transmitted bits. Accordingly, $\mu_{uv}^{k,r}, R_{uv}^{k,r}, Q_{uv}^{k,r}, D_{uv}^{k,r}$ become independent of r, and we simplify the notation by writing them as $\mu_{uv}^k, R_{uv}^k, Q_{uv}^k, D_{uv}^k$, respectively. As previously stated, a queue Q_{uv}^k follows an M/M/1 model and a well-known closed-form solution exists for the sojourn time, given by the inverse of the difference

TABLE I: Notation for the variables and parameters, alongside their measurement unit.

Symbol	Unit	Description
Input Parameters		
Λ^ϕ	[request/s]	Request arrival rate of service ϕ
$R_{uv}^{k,r}$	[resource/request]	Resource-r requirement of k on (u, v)
L^k	[s]	Maximum admissible latency of k
M_{uv}^r	[resource/s]	Resource-r capacity on (u, v)
c_{uv}^r	[cost/resource/s]	Cost per unit of rate of operating r on (u, v)
Decision Variables		
f_{uv}^k	[Boolean]	Flow of commodity k on edge (u, v)
μ_{uv}^r	[resource/s]	Resource-r service rate on (u, v) following SR model
$\mu_{uv}^{k,r}$	[resource/s]	Resource-r service rate of k on (u, v) following GR model
Auxiliary Variables (resulting from decision variables)		
λ_{uv}^k	[request/s]	Request arrival rate of k on (u, v)
λ_{uv}	[request/s]	Request arrival rate on (u, v)
ν_{uv}^k	[request/s]	Request service rate of k on (u, v)
d_{uv}^k	[s]	Latency experienced by k on (u, v)
l^k	[s]	Production-to-consumption latency of k
l_T^k	[s]	Total cumulative latency of k
ρ_{uv}^r	$[0, 1] \in \mathbb{R}$	Utilization factor of resource r on (u, v) following the SR model
$\rho_{uv}^{k,r}$	$[0, 1] \in \mathbb{R}$	Utilization factor related to k of resource r on (u, v) following the GR model
F_{uv}	[Boolean]	Activation of edge (u, v)

between the request service rate and the request arrival rate. In our model, this is expressed as:

$$\mathbb{E}[D_{uv}^k] = \frac{1}{\nu_{uv}^k - \lambda_{uv}^k} = \frac{R_{uv}^k}{\mu_{uv}^k - f_{uv}^k \Lambda^{\phi(k)} R_{uv}^k} \tag{1}$$

where we applied the definitions of ν_{uv}^k and λ_{uv}^k.

2) Communication links, SR model. Let $(u, v) \in \mathcal{E}^a$ be a communication link modeled under the SR model.

We model communication links with a single resource type $r \in \mathcal{R}$. Accordingly $\mu_{uv}^r, R_{uv}^{k,r}, Q_{uv}^r, D_{uv}^{k,r}$ become independent of r, and we simplify the notation by writing them as $\mu_{uv}, R_{uv}^k, Q_{uv}, D_{uv}^k$, respectively. In the SR model, Q_{uv} follows an M/G/1 model. We start by explicitly defining D_{uv}^k as the sum of two random variables: W_{uv}, representing the queuing time, and X_{uv}^k, representing the service time of requests of commodity k on queue Q_{uv}. Our goal is to define and model the expected sojourn time $\mathbb{E}[D_{uv}^k]$:

$$\mathbb{E}[D_{uv}^k] = \mathbb{E}[W_{uv}] + \mathbb{E}[X_{uv}^k] \tag{2}$$

We note that the service time depends on the workload requested by k, while the waiting time is independent of the incoming commodities under the FCFS scheduling policy.

Regarding the service time $\mathbb{E}[X_{uv}^k]$, since we have assumed exponential service times given a single commodity, we have $X_{uv}^k \sim f_{X_{uv}^k}(x) = \text{Exp}\left(\nu_{uv}^k\right)$. Therefore we have:

$$\mathbb{E}[X_{uv}^k] = \frac{1}{\nu_{uv}^k} = \frac{R_{uv}^k}{\mu_{uv}} \tag{3}$$

$$\mathbb{E}[(X_{uv}^k)^2] = 2\left(\frac{1}{\nu_{uv}^k}\right)^2 = 2\left(\frac{R_{uv}^k}{\mu_{uv}}\right)^2 \tag{4}$$

To compute the waiting time $\mathbb{E}[W_{uv}]$, we apply the *Pollaczek–Khinchine formula* [20], [24] for the expected waiting time in an M/G/1 queue:

$$\mathbb{E}[W_{uv}] = \frac{\lambda_{uv}\mathbb{E}[X_{uv}^2]}{2(1 - \rho_{uv})} \tag{5}$$

where the random variable X_{uv} represents the service time of a generic request on queue Q_{uv}, regardless of the specific commodity being served. To complete (5), we first compute the second moment of the service time, $\mathbb{E}[X_{uv}^2]$, and then the utilization factor, ρ_{uv}. The probability density function and the expected value of X_{uv} is given by:

$$X_{uv} \sim f_{X_{uv}}(x) = \sum_{k \in \mathcal{K}} \frac{\lambda_{uv}^k}{\lambda_{uv}} f_{X_{uv}^k}(x) \tag{6}$$

$$\mathbb{E}[X_{uv}] = \sum_{k \in \mathcal{K}} \frac{\lambda_{uv}^k}{\lambda_{uv}} \mathbb{E}[X_{uv}^k] \tag{7}$$

where the term $\frac{\lambda_{uv}^k}{\lambda_{uv}}$ represents the probability that a randomly selected request in queue Q_{uv} belongs to commodity k, thereby defining the contribution of $f_{X_{uv}^k}(x)$ to the overall service time distribution. To compute the second moment of the service time, we proceed as follows:

$$\mathbb{E}[(X_{uv})^2] = \int_0^{+\infty} x^2 \cdot f_{X_{uv}}(x)dx = \tag{8}$$

$$= \sum_{k \in \mathcal{K}} \frac{\lambda_{uv}^k}{\lambda_{uv}} \int_0^{+\infty} x^2 \cdot f_{X_{uv}^k}(x) =$$

$$= \sum_{k \in \mathcal{K}} \frac{\lambda_{uv}^k}{\lambda_{uv}} \mathbb{E}[(X_{uv}^k)^2]$$

Now, using (4) and the definition of λ_{uv}^k we have:

$$\mathbb{E}[(X_{uv})^2] = 2 \sum_{k \in \mathcal{K}} \frac{f_{uv}^k \Lambda^{\phi(k)}}{\lambda_{uv}} \left(\frac{R_{uv}^k}{\mu_{uv}}\right)^2 \tag{9}$$

Note that (9) captures the contribution of each individual commodity to the second moment of the service time, thereby enabling accurate modeling of queuing behavior. This formulation, which incorporates the flow variables f_{uv}^k, reflects whether a commodity is actively routed over the link and accounts for its precise impact on the congestion.

We now turn to the *utilization factor* ρ_{uv}, which, along with $\mathbb{E}[X_{uv}^2]$, completes the Pollaczek-Khinchine expression for the queuing time. As discussed earlier, since each communication link is associated with a single resource type $r \in R$, we simplify the notation by writing ρ_{uv}^r as ρ_{uv}.

To compute ρ_{uv}, we begin by defining the *per-commodity utilization* $\rho_{uv}^k = \lambda_{uv}^k/\nu_{uv}^k$, which represents the fraction of time queue Q_{uv} is busy serving commodity k. The total utilization of the queue is then the sum over all commodities:

$$\rho_{uv} = \sum_{k \in \mathcal{K}} \rho_{uv}^k.$$

Substituting the expressions for λ_{uv}^k and ν_{uv}^k yields:

$$\rho_{uv} = \sum_{k \in \mathcal{K}} \frac{\lambda_{uv}^k}{\nu_{uv}^k} = \frac{1}{\mu_{uv}} \sum_{k \in \mathcal{K}} f_{uv}^k \Lambda^{\phi(k)} R_{uv}^k. \tag{10}$$

This expression offers a clear physical interpretation: ρ_{uv} represents the ratio between the *total workload* imposed on the link by all active commodities and the *allocated resources*, expressed by the service rate μ_{uv}. It captures how congestion builds up based on both the intensity of incoming traffic and the per-commodity resource demands.

Combining (9) and (10) into (5), the final expected sojourn time of commodity k on queue Q_{uv} is:

$$\mathbb{E}[D_{uv}^k] = \mathbb{E}[W_{uv}] + \mathbb{E}[X_{uv}^k]$$

$$= \frac{\sum_{j \in \mathcal{K}} f_{uv}^j \Lambda^{\phi(j)} (R_{uv}^j)^2}{\mu_{uv}\left(\mu_{uv} - \sum_{j \in \mathcal{K}} f_{uv}^j \Lambda^{\phi(j)} R_{uv}^j\right)} + \frac{R_{uv}^k}{\mu_{uv}} \tag{11}$$

The derived formula represents the expected sojourn time $\mathbb{E}[D_{uv}^k]$ of a commodity k being routed into queue communication link (u, v) following the SR model.

3) Computation links, GR **model.** Let $(u, v) \in \mathcal{E}^a$ be a computation link, where $v = p_i$, modeled under the GR framework. We now derive the sojourn time formulation for computation links—and more generally, for commodities that require multiple resource types on the same link. To begin, we express the expected sojourn time for a commodity k with respect to each individual resource $r \in \mathcal{R}$ using the M/M/1 delay model:

$$\mathbb{E}[D_{uv}^{k,r}] = \frac{R_{uv}^{k,r}}{\mu_{uv}^{k,r} - f_{uv}^k \Lambda^{\phi(k)} R_{uv}^{k,r}} \tag{12}$$

Since a commodity may require service from multiple resources on the same node, its overall sojourn time is determined by the slowest (i.e., most congested) resource. Therefore, the total expected sojourn time is lower bounded by:

$$\mathbb{E}\left[D_{uv}^k\right] = \mathbb{E}\left[\max_{r \in \mathcal{R}} D_{uv}^{k,r}\right] \geq \max_{r \in \mathcal{R}} \mathbb{E}\left[D_{uv}^{k,r}\right] \tag{13}$$

where the inequality follows from Jensen's inequality [25], given that the max function is convex.

4) Computation links, SR **model.** Let $(u, v) \in \mathcal{E}^a$ be a computation link, with $v = p_i$, modeled following the SR model. In line with the approach used for the GR case, we express the average sojourn time of a SR link for each resource $r \in \mathcal{R}$ as follows:

$$\mathbb{E}[D_{uv}^{k,r}] = \frac{\sum_{j \in \mathcal{K}} f_{uv}^j \Lambda^{\phi(j)} (R_{uv}^{j,r})^2}{\mu_{uv}^r\left(\mu_{uv}^r - \sum_{j \in \mathcal{K}} f_{uv}^j \Lambda^{\phi(j)} R_{uv}^{j,r}\right)} + \frac{R_{uv}^{k,r}}{\mu_{uv}^r} \tag{14}$$

To obtain an overall estimate of the sojourn time, we again apply the maximization over resources:

$$\mathbb{E}\left[D_{uv}^k\right] \geq \max_{r \in \mathcal{R}} \mathbb{E}\left[D_{uv}^{k,r}\right] \tag{15}$$

Delay Constraints and Optimization Challenges: Together, Equations (1), (11), (13) and (15) give different formulations of the sojourn time $D_{uv}^{k,r}$ depending on whether (u, v) is a communication or computation link and whether it follows the GR or the SR model. We aim to constrain the various formulations of the sojourn time within a maximum limit by summing the sojourn times across all queues that a commodity traverses. However, the expression of the sojourn

time for the GR model is non-convex both in f_{uv}^k, μ_{uv}^r and $\mu_{uv}^{k,r}$, making direct optimization challenging. In Sec. IV we formulate the problem and then, in Sec. V, we will address the optimization by introducing suitable approximations and developing efficient approximation algorithms.

IV. PROBLEM FORMULATION

In this section, we describe the problem formulation. Our goal is to minimize a cost function that depends on service placement, routing, and resource allocation, subject to latency and capacity constraints. We formulate problem $\mathcal{P}$ as follows:

$$\min_{F_{uv}, \mu_{uv}^r, \mu_{uv}^{k,r}} \sum_{(u,v) \in \mathcal{E}^{a,\text{SR}}} \sum_{r \in \mathcal{R}} F_{uv} \mu_{uv}^r c_{uv}^r + \qquad (\mathcal{P})$$

$$+ \sum_{(u,v) \in \mathcal{E}^{a,\text{GR}}} \sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}} F_{uv} \mu_{uv}^{k,r} c_{uv}^r \text{, s.t.:}$$

$(a_1) \sum_{v \in N^-(u)} f_{vu}^k = \sum_{v \in N^+(u)} f_{uv}^k \qquad \forall u \in \mathcal{V}, k \in \mathcal{K}$

$(a_2) f_{pu}^k = f_{up}^l \qquad \forall (u,p) \in \mathcal{E}^a, k \in \mathcal{K}, l \in \mathcal{X}(k)$

$(a_3) f_{su}^k = 0 \qquad \forall k \in \mathcal{K}, u \neq s(k)$

$(a_4) f_{ud}^k = 1 \qquad \forall k \in \mathcal{K}^d, u = d(k)$

$(a_5) F_{uv} \geq f_{uv}^k \qquad \forall k \in \mathcal{K}, (u,v) \in \mathcal{E}^a$

$(b_1) d_{uv}^k \geq \mathbb{E}[D_{uv}^{k,r}(\boldsymbol{f}, \boldsymbol{\mu})] \qquad \forall (u,v) \in \mathcal{E}^a, k \in \mathcal{K}, r \in \mathcal{R}$

$(b_2) l^k = \sum_{(u,v) \in \mathcal{E}^a} f_{uv}^k d_{uv}^k \qquad \forall k \in \mathcal{K}$

$(b_3) l_T^k \geq l^k + l_T^j \qquad \forall k \in \mathcal{K} \setminus \mathcal{K}^s, j \in \mathcal{X}(k)$

$(b_4) l_T^k = l^k \qquad \forall k \in \mathcal{K}^s$

$(b_5) l_T^k \leq L^k \qquad \forall k \in \mathcal{K}^d$

$(c_1) \mu_{uv}^r > \sum_{k \in \mathcal{K}} f_{uv}^k \Lambda^{\phi(k)} R_{uv}^{k,r} \qquad \forall (u,v) \in \mathcal{E}^{a,\text{SR}}, r \in \mathcal{R}$

$(c_2) \mu_{uv}^{k,r} > f_{uv}^k \Lambda^{\phi(k)} R_{uv}^{k,r} \qquad \forall (u,v) \in \mathcal{E}^{a,\text{GR}}, k \in \mathcal{K}, r \in \mathcal{R}$

$(e_1) \mu_{uv}^r \leq M_{uv}^r \qquad \forall (u,v) \in \mathcal{E}^{a,\text{SR}}, r \in \mathcal{R}$

$(e_2) \sum_{k \in \mathcal{K}} \mu_{uv}^{k,r} \leq M_{uv}^r \qquad \forall (u,v) \in \mathcal{E}^{a,\text{GR}}, r \in \mathcal{R}$

$(e_3) f_{uv}^k \in \{0,1\} \qquad \forall (u,v) \in \mathcal{E}^a, k \in \mathcal{K}$

$(e_4) \mu_{uv}^r, \mu_{uv}^{k,r} \geq 0 \qquad \forall (u,v) \in \mathcal{E}^a, k \in \mathcal{K}, r \in \mathcal{R}$

The objective function to minimize is the total operational cost. In our notation, c_{uv}^r represents the cost per unit of resource-r service rate allocated on link $(u,v) \in \mathcal{E}^a$. The objective function is composed of two terms, accounting for the links that follow the SR and GR model respectively. The auxiliary variable F_{uv} takes the value 1 if at least one commodity traverses link (u,v), meaning the link needs to be activated, and 0 otherwise. Therefore, our cost model accounts for the links used in the augmented graph and the resource allocated to each of them. This cost model is particularly suitable to cloud on-demand services, like Amazon AWS or similar, where costs are influenced by both the geographical location of processing and the computational resources assigned to the VMs executing the operations.

The variables to be optimized are the flow variables and resource allocation variables. The flow variables are represented as $f_{uv}^k, (u,v) \in \mathcal{E}^a, k \in \mathcal{K}$. The resource allocation variables are represented with resource-r service rates $\mu_{uv}^{k,r}, (u,v) \in \mathcal{E}^{a,\text{GR}}, k \in \mathcal{K}, r \in \mathcal{R}$ in case a link is modeled with the GR model, requiring separate guaranteed resources for each commodity, and $\mu_{uv}^r, (u,v) \in \mathcal{E}^{a,\text{SR}}, r \in \mathcal{R}$ in case a link is modeled with the SR model, where a resource is shared across commodities.

Equations $(a_1) - (a_5)$ are flow conservation constraints. In particular, (a_1) ensures that the flow of a given commodity remains constant as it traverses a network link between two nodes, and (a_2) ensures that the flow consumed as input to create a new commodity is fully utilized in producing the new commodity for all its input components. This constraint preserves the DAG relationship between commodities. In the equation, we used the notation f_{up}^k indicating the flow passing through network node $u \in \mathcal{V}$ and its associated computation node in the augmented graph. Equation (a_3) ensures that no source node generates flow unless it is the node designated to generate the flow as specified by the problem's input. Similarly, (a_4) requires that all the flow must be directed to the designated destination node. Finally, constraint (a_5) deals with the F_{uv} indicator variable, assuming value 1 if link (u,v) is utilized, 0 otherwise.

Equations $(b_1) - (b_5)$ are delay constraints. Constraint (b_1) performs a maximization and bounds the auxiliary variable d_{uv}^k to the sojourn time of a request of commodity k on link (u,v). We indicate with $\boldsymbol{f}, \boldsymbol{\mu}$ the vectors containing f_{uv}^k, μ_{uv}^r variables. In the constraint, we make explicit the dependence of the average delay on $\boldsymbol{f}, \boldsymbol{\mu}$. We remark that the formulation of the average delay differs depending on whether (u,v) is a processing or a communication link, and on whether it follows the GR or SR model. Constraint, (b_2) defines l^k as the expected production-to-consumption delay experienced by commodity k as the summation of the sojourn times across all the queues it traverses. Equation (b_3) defines l_T^l which is the total cumulative latency of commodity k, taking into consideration the delay of input commodities $j \in \mathcal{X}(k)$. Then, equation (b_4) sets the total cumulative latency for source commodities to l^k, as they are not generated by any other commodity. Constraint (b_5) bounds the total cumulative latency to the maximum allowable end-to-end latency L^k, which is provided as an input to the problem.

Continuing, $(c_1) - (c_2)$ are queue stability bounds ensuring that the resource-r service rate is always greater than the incoming workload arrival rate. Finally, $(e_1) - (e_2)$ bound the allocated rates to be not greater than the system capacity, and $(e_3) - (e_4)$ define the domain of the variables.

We have presented the problem formulation along with all relevant constraints. Depending on whether an edge is modeled using the GR or SR approach, slightly different constraints are applied. The choice between the two ultimately depends on the specific problem setting and which model better reflects the physical characteristics of the underlying resource. In the case of the GR model for example, we have guaranteed and reserved resources per commodity, there is no contention among different commodities, and the delay experienced by one does not affect the delay of others. In contrast, the SR model assumes that the delays incurred by one

commodity affect the delay of all other commodities sharing the same resource. In this case, all commodities routed over the same link share the medium, and their delays are mutually dependent. Ultimately, the decision to model a link using the GR or SR model depends on the specific nature of the link and the characteristics of the environment. The proposed model is general and we presented different formulations of the average commodity sojourn time on a link following either the GR or SR model. Ultimately, the operator, who has knowledge of the application, the environment, and the network, must decide whether to model a link using the SR or GR model based on which approach best fits the scenario.

The presented formulation of problem $\mathcal{P}$ cannot be directly optimized for two main reasons. First, the integer domain of f_{uv}^k makes $\mathcal{P}$ a mixed integer program, which is NP-hard. Secondly, the calculation of $\mathbb{E}[D_{uv}^k]$ is non-convex in the SR model, as in Equations (11) and (14), due to the interdependent delays, even considering the linear relaxation of $\mathcal{P}$. Thus, in the next section, we introduce an algorithmic framework designed to find an approximated solution of the problem.

V. Algorithmic Solution

In this section, we present the key steps involved in addressing the optimization problem. The main issue in approaching the problem are Equations (11) and (14), whose non-linearity and non-convexity make direct optimization intractable.

To overcome this, we first introduce a tractable upper bound to the sojourn time by bounding the utilization factor of a queue. This step leads to a *bi-convex formulation*: the problem is not jointly convex, but it is convex in each block of variables when the other is fixed. We therefore decompose it into two convex sub-problems and solve them iteratively, optimizing f_{uv}^k and μ_{uv}^r in turn. Finally, we propose an approximation algorithm that alternately optimizes these sub-problems.

A. Delay upper bound

To make the derivation of the upper bound explicit, we apply the well-known ε-safety method to bound the utilization factor, obtaining a delay formulation that can be incorporated into our algorithm. Specifically, we bound the utilization factor of queue Q_{uv}^r as follows:

$$\rho_{uv}^r \leq 1 - \varepsilon_{uv}^r, \quad \varepsilon_{uv}^r \in [0,1], \forall r \in \mathcal{R} \tag{16}$$

for a sufficiently small ε_{uv}^r. This allows us to add an ε-safety margin on constraint (c_2) as follows:

$$(1 - \varepsilon_{uv}^r)\mu_{uv}^r \geq \sum_{k \in \mathcal{K}} f_{uv}^k \Lambda^{\phi(k)} R_{uv}^{k,r} \tag{17}$$

It is well known that this method ensures that the system operates below its capacity avoiding potential instability. We now find the sojourn time when the system is at its maximum permitted load, applying equation (17) into (11) (same applies for (14)). In this way we find an upper bound for the sojourn time, defined as $\mathbb{E}[\bar{D}_{uv}^{k,r}(\boldsymbol{f}, \boldsymbol{\mu}, \boldsymbol{\varepsilon})]$:

$$\mathbb{E}[D_{uv}^{k,r}(\boldsymbol{f}, \boldsymbol{\mu})] \leq \mathbb{E}[\bar{D}_{uv}^{k,r}(\boldsymbol{f}, \boldsymbol{\mu}, \boldsymbol{\varepsilon})] = \tag{18}$$
$$= \frac{\sum_{j \in \mathcal{K}} f_{uv}^j \Lambda^{\phi(k)} (R_{uv}^{j,r})^2}{\varepsilon_{uv}^r (\mu_{uv}^r)^2} + \frac{R_{uv}^{k,r}}{\mu_{uv}^r}$$

We observe that the previous formulation is convex in f_{uv}^k if we fix μ_{uv}^r and viceversa is convex in μ_{uv}^r if we fix f_{uv}^k. In the optimization problem, we use this upper bound as the delay metric to be minimized. In other words, the system is optimized under its maximum load. This approach is inherently suboptimal, since the solution may over-allocate resources in order to satisfy the latency constraints. However, the parameter ε_{uv}^r is optimized iteratively by the approximation algorithm discussed in Section V-C, which allows the solution to approach near-optimal performance.

B. Problem decomposition and convexification

In this section we decompose problem $\mathcal{P}$ into two sub-problems, namely $\mathcal{P}_1(\boldsymbol{\mu}, \boldsymbol{\varepsilon}, i)$ and $\mathcal{P}_2(\boldsymbol{f}, \boldsymbol{\varepsilon}, i)$. In $\mathcal{P}_1$ we optimize $\boldsymbol{f}$ variables and maintain $\boldsymbol{\mu}$ fixed, while in $\mathcal{P}_2$ we optimize $\boldsymbol{\mu}$ variables and maintain $\boldsymbol{f}$ fixed. The problems are used in the iterative algorithm presented in Section V-C, and i indicates the iteration number. We indicate as $f_{uv}^k(i), \mu_{uv}^r(i), \mu_{uv}^{k,r}(i), \varepsilon_{uv}^r(i)$ the values of the variables at iteration i. We formulate $\mathcal{P}_1(\boldsymbol{\mu}, \boldsymbol{\varepsilon}, i)$ as follows:

$$\min_{F_{uv}} \sum_{(u,v) \in \mathcal{E}^{a,\text{SR}}} \sum_{r \in \mathcal{R}} \mu_{uv}^r(i) F_{uv} c_{uv}^r + \qquad (\mathcal{P}_1(\boldsymbol{\mu}, \boldsymbol{\varepsilon}, i))$$
$$+ \sum_{(u,v) \in \mathcal{E}^{a,\text{GR}}} \sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}} \mu_{uv}^{k,r}(i) F_{uv} c_{uv}^r \text{, s.t.:}$$

$$\frac{1}{1 - \varepsilon_{uv}^r(i)} \sum_{k \in \mathcal{K}} \hat{f}_{uv}^k \Lambda^{\phi(k)} R_{uv}^{k,r} \leq \mu_{uv}^r(i) | \forall (u,v) \in \mathcal{E}^a, r \in \mathcal{R}$$

$$\sum_{v \in N^-(u)} \hat{f}_{vu}^k = \sum_{v \in N^+(u)} \hat{f}_{uv}^k \quad \forall u \in \mathcal{V}, k \in \mathcal{K}$$

$$\hat{f}_{pu}^k = \hat{f}_{up}^l \qquad \forall u \in \mathcal{V}, k \in \mathcal{K}, l \in \mathcal{X}(k)$$
$$\hat{f}_{su}^k = 0 \qquad \forall k \in \mathcal{K}, u \neq s(k)$$
$$\hat{f}_{ud}^k = 1 \qquad \forall k \in \mathcal{K}^d, u = d(k)$$
$$F_{uv} \geq \hat{f}_{uv}^k \qquad \forall k \in \mathcal{K}, (u,v) \in \mathcal{E}^a$$
$$\bar{d}_{uv}^k \geq \mathbb{E}\left[\bar{D}_{uv}^{k,r}(\boldsymbol{f}, \boldsymbol{\mu}(i), \boldsymbol{\varepsilon}(i))\right] \quad \forall (u,v) \in \mathcal{E}^a, k \in \mathcal{K}, r \in \mathcal{R}$$
$$l^k = \sum_{(u,v) \in \mathcal{E}^a} \hat{f}_{uv}^k \bar{d}_{uv}^k \qquad \forall k \in \mathcal{K}$$
$$l_T^k = l^k \qquad \forall k \in \mathcal{K}^s$$
$$l_T^k \geq l^k + l_T^j \qquad \forall k \in \mathcal{K} \setminus \mathcal{K}^s, j \in \mathcal{X}(k)$$
$$l_T^k \leq L^k \qquad \forall k \in \mathcal{K}^d$$
$$\hat{f}_{uv}^k \in [0,1] \qquad \forall (u,v) \in \mathcal{E}^a, k \in \mathcal{K}$$

Here, we have incorporated the utilization factor bound from Sec. V-A into the queue stability constraint, ensuring that the delay formulation remains tractable. Also, we have introduced the variables $\hat{f}_{uv}^k$ which represent the continuous relaxation for the original problem. Nevertheless, the problem remains non-convex: by expanding the definition of l^k, we obtain bilinear terms of the form $\hat{f}_{uv}^k \cdot \hat{f}_{uv}^j$. To handle this, we adopt the method described in [26], where bilinear terms $g(\hat{f}_{uv}^k, \hat{f}_{uv}^j) = \hat{f}_{uv}^k \cdot \hat{f}_{uv}^j$ are convexified by linearizing their concave component. Following [26], for any given $y_1, y_2 \in \mathbb{R}$, the concave part can be approximated using the values $\hat{f}_{uv}^k(i-1), \hat{f}_{uv}^j(i-1)$

from the previous iteration. This procedure ensures that at each iteration i, problem $\mathcal{P}_1(\boldsymbol{\mu}, \boldsymbol{\varepsilon}, i)$ becomes convex in the flow variables $\hat{f}_{uv}^k$ and can therefore be solved efficiently.

We now formulate $\mathcal{P}_2$ as follows:

$$\min_{\mu_{uv}^r, \mu_{uv}^{k,r}} \sum_{(u,v)\in\mathcal{E}^{a,\mathrm{SR}}} \sum_{r\in\mathcal{R}} F_{uv}(i)\mu_{uv}^r c_{uv}^r + \qquad (\mathcal{P}_2(\boldsymbol{f}, \boldsymbol{\varepsilon}, i))$$

$$+ \sum_{(u,v)\in\mathcal{E}^{a,\mathrm{GR}}} \sum_{k\in\mathcal{K}} \sum_{r\in\mathcal{R}} F_{uv}(i)\mu_{uv}^{k,r} c_{uv}^r, \text{ s.t.:}$$

$$\mu_{uv}^r \geq \frac{\sum_{k\in\mathcal{K}} f_{uv}^k(i)\Lambda^{\phi(k)} R_{uv}^{k,r}}{1-\varepsilon_{uv}^r(i)} \quad \forall(u,v)\in\mathcal{E}^a, r\in\mathcal{R}$$

$$\mu_{uv}^{k,r} \geq \frac{f_{uv}^k(i)\Lambda^{\phi(k)} R_{uv}^{k,r}}{1-\varepsilon_{uv}^r(i)} \qquad \forall(u,v)\in\mathcal{E}^a, k\in\mathcal{K}, r\in\mathcal{R}$$

$$\bar{d}_{uv}^k \geq \mathbb{E}\left[\bar{D}_{uv}^{k,r}(\boldsymbol{f}(i), \boldsymbol{\mu}, \boldsymbol{\varepsilon}(i))\right] | \forall(u,v)\in\mathcal{E}^a, k\in\mathcal{K}, r\in\mathcal{R}$$

$$l^k = \sum_{(u,v)\in\mathcal{E}^a} f_{uv}^k(i)\,\bar{d}_{uv}^k \qquad \forall k\in\mathcal{K}$$

$$l_T^k = l^k \qquad \forall k\in\mathcal{K}^s$$

$$l_T^k \geq l^k + l_T^j \qquad \forall k\in\mathcal{K}\setminus\mathcal{K}^s, j\in\mathcal{X}(k)$$

$$l_T^k \leq L^k \qquad \forall k\in\mathcal{K}^d$$

$$\mu_{uv}^r \leq M_{uv}^r \qquad \forall(u,v)\in\mathcal{E}^a, r\in\mathcal{R}$$

$$\sum_{k\in\mathcal{K}} \mu_{uv}^{k,r} \leq M_{uv}^r \qquad \forall(u,v)\in\mathcal{E}^a, r\in\mathcal{R}$$

$$\mu_{uv}^r, \mu_{uv}^{k,r} \geq 0 \qquad \forall(u,v)\in\mathcal{E}^a, k\in\mathcal{K}, r\in\mathcal{R}$$

where we have used $F_{uv} = \max_{k\in\mathcal{K}} \hat{f}_{uv}^k$. We note that $\mathcal{P}_2$ is convex for resource allocation variables μ_{uv}^r.

C. Approximation algorithm

We now define the SPARQ algorithm that optimizes $\mathcal{P}_1$ and $\mathcal{P}_2$ in an alternating and iterative manner, converging to a solution in the continuous domain of the original problem described in Section IV. The algorithm is shown in Alg. 1 and it's an expansion of the NOVA algorithm presented in [26]. In the algorithm, we indicate $\bar{\boldsymbol{f}}(i) = \mathcal{P}_1(\boldsymbol{\mu}, \boldsymbol{\varepsilon}, i)$ to be the solution of $\mathcal{P}_1$ and $\bar{\boldsymbol{\mu}}(i) = \mathcal{P}_2(\bar{\boldsymbol{f}}, \boldsymbol{\varepsilon}, i)$ to be the solution of $\mathcal{P}_2$. The steps of the algorithm are as follows:

Algorithm 1 Service Placement, Resource Allocation and Routing with Queue-Aware Delays (SPARQ)

1: Initialize $\boldsymbol{\mu}(0) \leftarrow M, \boldsymbol{\varepsilon}(0) \leftarrow \frac{3}{4}$
2: **for** $i = 1$ to maximum iterations or convergence **do**
3:     $\bar{\boldsymbol{f}}(i) \leftarrow \mathcal{P}_1(\boldsymbol{\mu}, \boldsymbol{\varepsilon}, i-1)$
4:     $\bar{\boldsymbol{\mu}}(i) \leftarrow \mathcal{P}_2(\bar{\boldsymbol{f}}, \boldsymbol{\varepsilon}, i-1)$
5:     $\hat{\boldsymbol{f}}(i+1) \leftarrow \hat{\boldsymbol{f}}(i) + \gamma_i(\bar{\boldsymbol{f}}(i) - \hat{\boldsymbol{f}}(i))$
6:     $\boldsymbol{\mu}(i+1) \leftarrow \boldsymbol{\mu}(i) + \gamma_i(\bar{\boldsymbol{\mu}}(i) - \boldsymbol{\mu}(i))$
7:     Compute $\boldsymbol{\rho}(i)$ with $\bar{\boldsymbol{f}}(i), \bar{\boldsymbol{\mu}}(i)$ following Eq. (10)
8:     $\boldsymbol{\varepsilon}(i+1) \leftarrow \gamma_i\boldsymbol{\varepsilon}(i) + (1-\gamma_i)(1-\boldsymbol{\rho}(i))$
9:     $i \leftarrow i+1$
10: **end for**
11: $\boldsymbol{f} \leftarrow$ Decomposition and Rounding IDAGO of $\hat{\boldsymbol{f}}(i+1)$

The algorithm starts by initializing the resource allocation variables to their capacity M, indicating the vector containing

M_{uv}^r values. It then proceeds by solving sub-problem $\mathcal{P}_1$ yielding the flow values $\bar{\boldsymbol{f}}(i)$. Next, sub-problem $\mathcal{P}_2$ is solved using $\bar{\boldsymbol{f}}(i)$ as input flow, resulting in $\bar{\boldsymbol{\mu}}(i)$, which represents the minimum service rate required to sustain the flow $\bar{\boldsymbol{f}}(i)$ under the latency constraints. Then, we use a factor γ_i to gradually update the variables. We use classical diminishing step-size rules for γ_i, which satisfy the conditions in [26]. Such approach guarantees asymptotic convergence of the algorithm to stationary points of Problem $\mathcal{P}$. Continuing, the algorithm computes the vector $\boldsymbol{\rho}(i)$, which contains the observed utilization factor ρ_{uv}^r with the instantaneous $\bar{\boldsymbol{f}}(i), \bar{\boldsymbol{\mu}}(i)$ values, computed following Eq. (10). Then, the vector $\boldsymbol{\varepsilon}(i)$ is updated as a function of the computed utilization $\boldsymbol{\rho}(i)$, and the procedure is repeated until convergence. This update adaptively tunes the safety margins to reflect the current system load, thereby improving the tightness of the utilization factor and delay bound.

In general, the algorithm iterates by progressively selecting smaller values of $\boldsymbol{\mu}$, optimizing the flow at each step. With each iteration, it moves closer to the optimal solution, refining both the flow and the resource allocation variables. We observe that the algorithm never assigns values to the variables $\boldsymbol{\mu}$ that exceed their respective upper limits M: initially, $\boldsymbol{\mu}$ is set to their maximum allowable values, i.e., $\boldsymbol{\mu}(0) = M$. In the subsequent steps, $\mathcal{P}_1$ is solved. Since the initial values of $\boldsymbol{\mu}(0)$ are at their upper bounds, the solution $\bar{\boldsymbol{f}}(i)$ is obtained using the maximum possible capacity of the system. Next, $\mathcal{P}_2$ is solved to determine the minimal values of $\bar{\boldsymbol{\mu}}(i)$ that are sufficient to support the flow $\bar{\boldsymbol{f}}(i)$. Importantly, the flow $\bar{\boldsymbol{f}}(i)$ was already supported by the maximum capacity found in the previous step. Thus, either the problem is unfeasible or the solution of $\mathcal{P}_2$ must produce $\bar{\boldsymbol{\mu}}(i)$ values that are no greater than the current $\boldsymbol{\mu}(i-1)$ values.

Also, we note that the algorithm refines the values of $\boldsymbol{\varepsilon}(i)$ throughout the iterations. As the algorithm progresses, $\boldsymbol{\varepsilon}(i)$ is iteratively updated towards the real and observed $1 - \boldsymbol{\rho}(i)$, computed using the current values $\bar{\boldsymbol{f}}(i), \bar{\boldsymbol{\mu}}(i)$. This has the effect of making the upper bound $\mathbb{E}[\bar{D}_{uv}^{k,r}(\boldsymbol{f}, \boldsymbol{\mu}, \boldsymbol{\varepsilon})]$ converge to its real value $\mathbb{E}[D_{uv}^{k,r}(\boldsymbol{f}, \boldsymbol{\mu}, \boldsymbol{\varepsilon})]$, whose formulation is non-convex.

We have presented an algorithm that decomposes and solves the optimization problem iteratively in the continuous domain. However, to obtain an admissible solution for $\mathcal{P}$, the flow variables are required to be integer. This necessitates the use of a rounding strategy. In this work, we adopt the same rounding technique used in IDAGO, a bi-criteria optimization algorithm originally introduced in [27]. IDAGO was developed to address multi-commodity flow placement and resource allocation, followed by integer rounding of flow variables on the same cloud-augmented graph. The IDAGO rounding technique is efficient, requiring the heavy convex optimization step to be executed only once. The IDAGO rounding procedure generates multiple candidate embeddings, where each embedding consists of a set of rounded flow variables that satisfy flow conservation. Probabilities are assigned to each candidate embedding, and the final embedding is selected through randomized sampling based on these probabilities. IDAGO comes with formal guarantees on the quality of the

solution, expressed in terms of its approximation ratio.

Eventually, SPARQ finds a solution $\mathcal{S}$ with integer f_{uv}^k variables which is returned as the final solution of problem $\mathcal{P}$. In the following section, we apply the whole modeling, decomposition and algorithm pipeline discussed in the previous sections to our experimental setup and simulations.
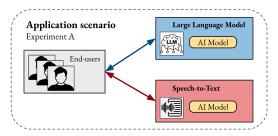
VI. RESULTS

In this section, we present the experimental setup and the results we obtained. In all experiments, we assessed our queue modeling approach against the private delay model in [27]. To strengthen the allocation given by the private delay model, rates $\mu_{uv}^r, \mu_{uv}^{k,r}$ are deliberately over-allocated by a constant $\alpha \geq 1$. We will observe that, no matter the multiplication constant α, the system with the private delay model produces either non-valid or sub-optimal solutions: either because the allocated resources are too low to satisfy the latency constraints or because over-allocation generates a sub-optimal and unnecessarily costly solution.

In all experiments, we apply SPARQ to find a solution for the placement, routing and resource allocation. We then use the values of the variables and compute values such as the latency and the objective cost.
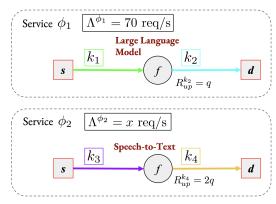
A. Experiment A

The first experiment involves a small network with a pool of users and two available AI services, reported in Fig. 6. The goal of this experiment is to determine the optimal computation placement, flow routing and resource allocation across the edge-cloud network. An overview of the application is reported in Fig. 6a. We modeled this experiment starting from the service graph, reported in Fig. 6b. We consider two AI services acting on the network: a Speech-to-text model and a Large Language Model, respectively indicated as ϕ_1, ϕ_2, In our experiments we modeled ϕ_2 to require double the amount of processing than ϕ_1, namely q. Given the AI and heavyweight nature of the services, both of them are modeled following the SR model. Both services have network node u as respective source and destination. In Fig. 6c is reported the augmented graph of the edge-cloud network. It consists of a user node u, a cloud computational node n and an edge computational node e, connected by routers. We modeled the cloud processing cost c_{pn} as a parameter c, and we set the edge processing cost to be $c_{pe} = 10c$. The cost coefficients are generic and tunable, allowing the model to reflect operator or application preferences. For instance, lowering edge costs favors edge processing, while higher values naturally bias the computation toward the cloud.
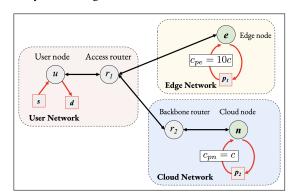
In this experiment, we fixed the request arrival rate of the first service Λ^{ϕ_1} and we increased the rate of the second service Λ^{ϕ_2}. The two output commodities k_2, k_4 are constrained to a maximum latency of $L^{k_2} = L^{k_4} = 100$ms. In Figure 7 is reported the measured latency of k_4. When the value of Λ^{ϕ_2} is low, the system performs all computations on node n, as it is cheaper to operate. However, as the rate Λ^{ϕ_2} increases beyond a certain threshold, node n can no longer handle the processing of both commodities, and the system is forced to

(a) Application overview of experiment A. A pool of users accesses different AI models to achieve different tasks.

(b) Service graph of experiment A, consisting of two services with only two commodities each, i.e. one input and one output commodity with a single function.

(c) Augmented graph of experiment A, consisting of a network with two computation nodes, having different computational costs.

Fig. 6: Experiment A application, augmented graph and service graph.

activate node e. The rate at which this transition occurs is indicated by the vertical red line in the figure. The reported latency is the real, measured, a-posteriori latency, which is *not* directly optimized by the algorithm since it is non-convex. When measured after the optimization algorithm has set the variables values, it remains very close to the boundary, hence to the optimality. We observe that the private delay model solution struggles to stay below the L^{k_4} limit, caused by the non-balanced distribution of requested service $R_{uv}^{k,r}$ on the same shared resource. For the latency to exceed L^{k_4}, the rates must be multiplied by a constant α greater than 1.4 approximately. However, such solutions are sub-optimal, being well below the latency boundary, resulting in a cost-inefficient
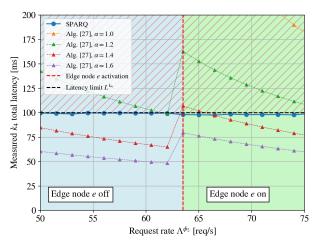
Fig. 7: Measured latency of output commodity k_4 on rate Λ^{ϕ_2} change for experiment A. The dashed black horizontal line is the maximum latency limit L^{k_4} and the vertical red line is the activation of processing of node e. The upper area of the graph, shaded with red diagonal lines, violates the latency constraints.
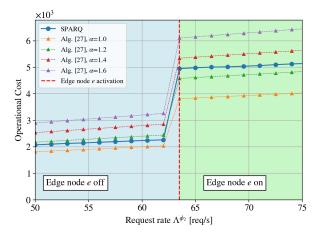

Fig. 8: Operational cost of experiment A on change of request rate Λ^{ϕ_2}.

allocation, as shown in the following.

Fig. 8 shows the behavior of the objective function. We remark that the objective function represents the operational cost of the placement, routing and resource allocation of the deployment. In the private delay model, the cost increases as the allocation multiplier α increases. Private delay model solutions with $\alpha \in \{1.4, 1.6\}$ achieve a higher cost compared with SPARQ. Solutions with $\alpha \in \{1.0, 1.2\}$ yield a lower cost than SPARQ but produce latencies exceeding the threshold L^{k_4} by a significant margin. The cost-efficiency of the solution can be observed in Fig. 9, presenting the trade-off curves of the different solutions on the cost-latency space. In this experiment, the constant latency limit bisects the plane into two areas which contain solutions with measured latency below and above 100ms respectively, the latter being unfeasible for the problem. Again, we observe that our solution lies close to the feasible boundary, indicating the scarce resource
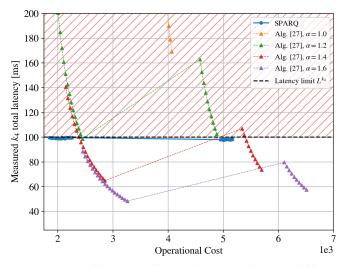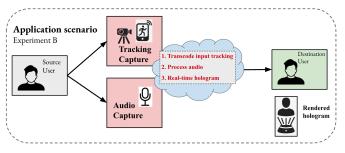

Fig. 9: Trade-off curves of Experiment A for the different solutions. With equal latency, our solution has the lowest operational cost which stays below the maximum latency limit L^{k_4}.

over-allocation carried out by our solution. From the trade-off curves we can observe that with equal latency we obtain the lowest cost compared to the other solutions.
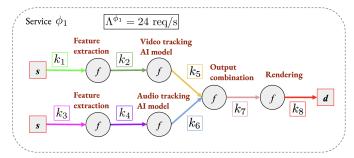
B. Experiment B

In this experiment, we investigate an application involving an Augmented Reality (AR) communication system. This application is based on the FANTASIA framework [28], which has already been implemented in other applications [29]. The modeling is shown in Fig. 10. Specifically, the overview of the application is shown in Fig. 10a. The system is designed as a provider of AR communication, where the user captures video, motion tracking, and audio. The application is responsible for managing raw data streams, including audio, video, and tracking information. Afterward, the system performs feature extraction on the captured data, passes it through AI models for various tasks such as recognition, reconstruction, and other processing purposes. The outputs from these models are then combined and rendered onto a final AR/VR device for the destination user, creating a real-time holographic experience. This is summarized in the application service graph in Fig. 10b. The commodities requiring process by AI models, are modeled following the SR model, while the others follow the GR model. The network graph in Fig. 10c shows the augmented graph of the system, consisting of multiple nodes and paths connecting the sources of the source user's streams to the destination user's node v. We observe that many nodes in the network possess computational capabilities, including those in edge networks close to users as well as cloud nodes. In particular, the destination node v also has processing capabilities that can be utilized to process commodities.
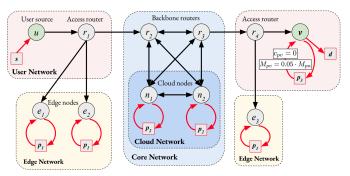
In this experiment, we analyze the optimal service allocation from the application provider's perspective, focusing on minimizing the economic cost. Here, the provider is faced with the decision of how to allocate computational tasks, having two options:

(a) Application overview of FANTASIA [29].

(b) Application service graph of FANTASIA. The commodity k_8 represents the final rendered data stream.

(c) Augmented graph of experiment B. The user equipment, represented by node v, has a computation node modeled with 0 associated cost and a capacity which is 5% of the capacity of the other nodes in the network.

Fig. 10: Experiment B application, service graph and network graph.

1) processing the video streams on edge and/or cloud resources incurring in operational costs;
2) offloading computations to the user's end-device, incurring in no operational costs, but considering that the user's equipment has limited computational capabilities.

In particular, the provider may decide whether to offload the final rendering operation, represented by commodity k_8, to the user's equipment, provided that latency and capacity constraints are met. We modeled this environment as follows. The process of producing the output commodity k_8 can be offloaded to v with an associated cost $c_{pv}^r = 0$ and v's processing capacity is constrained to 5% of the capacity of other computing nodes.

In this experiment, we varied the maximum allowed latency L^{k_8} and observed the system's response. Fig. 11 shows the total latency experienced by the application. As in the previ-
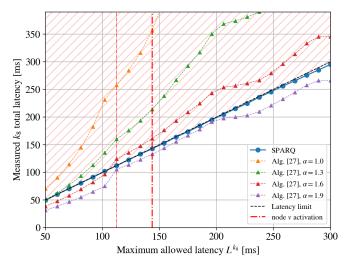

Fig. 11: Measured latency experienced by the application, subject to latency constraints.

ous experiment, the considered latency is the real, measured latency which is not directly optimized by the system. The black dashed line in Fig. 11 indicates the latency limit. An optimal solution should align with this boundary, and our solution consistently remains very close without exceeding it. In our solution, when L^{k_8} exceeds approximately 140ms, the system activates processing on user node v, indicated by the rightmost vertical red line in the figure. Below this threshold, the system recognizes that even fully utilizing the computational resources of node v, it would not meet the latency constraint. Consequently, node v is disabled, and the computation is shifted to other nodes. In contrast, the private delay model solutions are less conservative and continue to use node v for lower and more stringent values of L^{k_8}, indicated by the leftmost vertical red line in the figure, which results in exceeding the latency constraint.

Fig. 12 shows the cost associated to the solutions. Combining the results from figures 11, 12, it can be seen that solutions with $\alpha \in \{1.0, 1.3\}$ always exceed the threshold, violating the latency constraint. The solution with $\alpha = 1.6$ starts below the latency limit and costing more than our solution, and then violates the latency limit when it activates node v for the computation. Finally, the solution with $\alpha = 1.9$ stays always below the latency requirement but it allocates more resources respect to our solution, leading to sub-optimal solutions with unnecessarily low latency and higher costs.

Fig. 13 illustrates the trade-off in the cost-latency space for the different solutions. In this experiment, the maximum latency constraint is variable, meaning that it does not strictly divide the plane into feasible and infeasible regions as Experiment A. Nevertheless, the curves demonstrate that our proposed approach consistently achieves the lowest cost for a given latency, even though some solutions may fall outside the feasible range. Furthermore, the comparison curves are closely aligned, indicating that the efficiency of these alternative solutions is similar. In contrast, our solution exhibits a clear separation from the others, highlighting its higher efficiency by a notable margin.
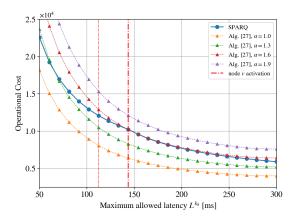
Fig. 12: Operational cost of the solution on the variation of the maximum allowed latency L^{k_5}.
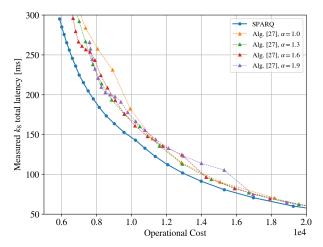

Fig. 13: Trade-off curves of Experiment B in the cost-latency space for the different solutions. Our solution attains the most efficient placement and resource allocation compared to the others.

VII. Conclusions

In this work, we tackle the optimization of placement, routing, and communication-computation resource allocation of AI-intensive applications over edge-cloud architectures. We propose a novel modeling approach that by combining two key resource models, Guaranteed-Resource (GR) and Shared-Resource (SR) models, provides a more accurate representation of how modern applications are executed on physical hardware, particularly in the context of resource-intensive AI workloads. The GR model ensures dedicated resources for each task, while the SR model accounts for the shared nature of certain resources, where delays are influenced by different tasks running concurrently.

To model the resulting intricate resource-delay dynamics, we introduce a queue-based methodology that uses a mixture of M/G/1 and M/M/1 queues to accurately capture the non-linear relationship between resource allocation and service delays.

Given the non-convexity of the resulting optimization problem, we develop a set of approximation and convexification techniques that decompose the problem into two tractable sub-

problems. The proposed iterative algorithm, termed SPARQ, jointly optimizes service placement, routing, and resource allocation under non-linear delays. Experimental results demonstrate the effectiveness of our approach over traditional methods. In particular, SPARQ produces a solution of lower cost, while maintaining latencies under the maximum threshold, even in the presence of unbalanced service requests.

To conclude, we remark that the generality of the resource-delay model presented in this paper makes it suitable for many emerging applications and computing environments. A direction for future work is to apply this model in production environments, such as Amazon AWS, for the orchestration of real agentic-AI applications.

References

[1] Y. Sun, Z. Chen, M. Tao, and H. Liu, "Communications, caching, and computing for mobile virtual reality: Modeling and tradeoff," *IEEE Transactions on Communications*, vol. 67, no. 11, pp. 7573–7586, 2019.

[2] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward low-latency and ultra-reliable virtual reality," *IEEE Network*, vol. 32, no. 2, pp. 78–84, 2018.

[3] DeepSeek-AI, "Deepseek-v3 technical report," 2024.

[4] M. Barcelo, A. Correa, J. Llorca, A. M. Tulino, J. L. Vicario, and A. Morell, "Iot-cloud service optimization in next generation smart environments," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 4077–4090, 2016.

[5] M. Michael, J. Llorca, and A. Tulino, "Approximation algorithms for the optimal distribution of real-time stream-processing services," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–7, 2019.

[6] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in mec networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1047–1060, 2020.

[7] K. Poularakis, J. Llorca, A. M. Tulino, and L. Tassiulas, "Approximation algorithms for data-intensive service chain embedding," in *ACM Mobihoc*, ACM, 2020.

[8] A. Mauro, A. M. Tulino, and J. Llorca, "End-to-end orchestration of nextg media services over the distributed compute continuum," *arXiv preprint arXiv:2407.08710*, 2024.

[9] J. Llorca and A. M. Tulino, "Cloud network flow: Understanding information flow in nextg cloud-integrated networks," *arXiv preprint*, 2024.

[10] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "Optimal dynamic cloud network control," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2118–2131, 2018.

[11] J. Zhang, A. Sinha, J. Llorca, A. M. Tulino, and E. Modiano, "Optimal control of distributed computing networks with mixed-cast traffic flows," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1760–1773, 2021.

[12] F. Pezone, S. Barbarossa, and P. Di Lorenzo, "Goal-oriented communication for edge learning based on the information bottleneck," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8832–8836, 2022.

[13] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Joint compute-caching-communication control for online data-intensive service delivery," *IEEE Transactions on Mobile Computing*, 2023.

[14] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Optimal multicast service chain control: Packet processing, routing, and duplication," in *ICC 2021 - IEEE International Conference on Communications*, pp. 1–7, 2021.

[15] M. Merluzzi, P. D. Lorenzo, S. Barbarossa, and V. Frascolla, "Dynamic computation offloading in multi-access edge computing via ultra-reliable and low-latency communications," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 6, pp. 342–356, 2020.

[16] G. C. Sankaran and K. M. Sivalingam, "Design and analysis of fast ip address-lookup schemes based on cooperation among routers," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pp. 330–339, 2020.

[17] R. Avazeh and N. Yazdani, "A new tcam architecture for ip routing with update complexity equal to o(1)," *Canadian Journal of Electrical and Computer Engineering*, vol. 43, no. 4, pp. 207–217, 2020.

[18] M. Polverini, J. Galán-Jiménez, F. G. Lavacca, A. Cianfrani, and V. Eramo, "A scalable and offloading-based traffic classification solution in nfv/sdn network architectures," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1445–1460, 2021.

[19] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "Vnf placement and resource allocation for the support of vertical services in 5g networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 433–446, 2019.

[20] A. Baiocchi, *Queueing Networks*, pp. 331–397. Wiley, 2020.

[21] M. Barcelo, J. Llorca, A. M. Tulino, and N. Raman, "The cloud service distribution problem in distributed cloud networks," in *2015 IEEE International Conference on Communications (ICC)*, pp. 344–350, 2015.

[22] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, O. Vinyals, J. W. Rae, and L. Sifre, "Training compute-optimal large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, (Red Hook, NY, USA), Curran Associates Inc., 2022.

[23] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020.

[24] M. Miyazawa, "A generalized pollaczek-khinchine formula for the gi/gi/1/k queue and its application to approximation," *Stochastic Models*, vol. 3, pp. 53–65, 1987.

[25] J. A. T. Thomas M. Cover, *Inequalities in Information Theory*, ch. 17, pp. 657–687. John Wiley & Sons, Ltd, 2005.

[26] G. Scutari, F. Facchinei, and L. Lampariello, "Parallel and distributed methods for constrained nonconvex optimization—part i: Theory," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1929–1944, 2017.

[27] A. Mauro, A. M. Tulino, and J. Llorca, "End-to-end orchestration of nextg media services over the distributed compute continuum," *arXiv preprint arXiv:2407.08710*, 2024.

[28] A. Origlia, F. Cutugno, A. Rodà, P. Cosi, and C. Zmarich, "Fantasia: a framework for advanced natural tools and applications in social, interactive approaches," *Multimedia Tools and Applications*, vol. 78, pp. 13613–13648, May 2019.

[29] A. Origlia, M. L. Chiacchio, M. Grazioso, and F. Cutugno, "Increasing visitors attention with introductory portal technology to complex cultural sites," *International Journal of Human-Computer Studies*, vol. 180, p. 103135, 2023.