






# Information Sharing in Multi-Tenant Metaverse via Intent-Driven Multicasting

Yu Qiu , Min Chen , *Fellow, IEEE*, Weifa Liang , *Senior Member, IEEE*,  
Lejun Ai , and Dusit Niyato , *Fellow, IEEE*

**Abstract**—A multi-tenant metaverse enables multiple users in a common virtual world to interact with each other online. Information sharing will occur when interactions between a user and the environment are multicast to other users by an interactive metaverse (IM) service. However, ineffective information-sharing strategies intensify competitions among users for limited resources in networks, and fail to interpret optimization intent prompts conveyed in high-level natural languages, ultimately diminishing user immersion. In this paper, we explore reliable information sharing in a multi-tenant metaverse with time-varying resource capacities and costs, where IM services are unreliable and alter the volumes of data processed by them, while the service provider dynamically adjusts global intent to minimize multicast delays and costs. To this end, we first formulate the information sharing problem as a Markov decision process and show its NP-hardness. Then, we propose a learning-based system GTP, which combines the proximal policy optimization reinforcement learning with feature extraction networks, including graph attention network and gated recurrent unit, and a Transformer encoder for multi-feature comparison to process a sequence of incoming multicast requests without the knowledge of future arrival information. The GTP operates through three modules: a deployer that allocates primary and backup IM services across the network to minimize a weighted goal of server computation costs and communication distances between users and services, an

intent extractor that dynamically infers provider intent conveyed in natural language, and a router that constructs on-demand multicast routing trees adhering to users, the provider, and network constraints. We finally conduct theoretical and empirical analysis on the proposed algorithms for the system. Experimental results show that the proposed algorithms are promising, and superior to their comparison baseline algorithms.

**Index Terms**—Multi-tenant, metaverse, intent network, multicast, information sharing, reliability.

## I. INTRODUCTION

### A. Background

**M**ULTI-TENANT metaverse is an immersive, ubiquitous, shared, and persistent cyberspace that hosts multiple virtual worlds, each tailored to specific scenarios such as industry, healthcare, and beyond [19]. Within a single scenario, multiple users interact with the shared world collectively through their digital avatars to enjoy their second life, where related data on these interactions is continuously transmitted through interactive metaverse (IM) services hosted on edge servers. Information sharing occurs when interactions between a user and the virtual surroundings are synchronized with other users via multicast routing [6]. Specifically, personal data is initially processed by the IM service, and then the calculated results are subsequently transmitted to other participating users, as shown in Fig. 1. Real-time information sharing ensures that these interactions are consistently updated and aligned across the entire metaverse. As a result, users experience a cohesive and dynamic metaverse scenario that allows them to collectively shape and enrich the environment, seamlessly blending physical and digital realities [17], [18].

The multi-tenant metaverse greatly enhances immersive experiences, but it also demands more automated network management from service providers. Intent-based networking [2], [11], as a promising paradigm, is integrated with the metaverse to address this need, which optimizes service provisioning based on high-level intents rather than dealing with intricate and low-level network commands. By automatically translating intents into commands that adjust underlying infrastructure resources, the metaverses enable service providers and users to steer network management more intelligently and succinctly, reducing the risk of human-induced configuration errors. For instance, a service provider might specify intent in natural language, such as “Provide low-latency multicasting for virtual

Received 9 March 2025; revised 7 July 2025; accepted 23 August 2025. Date of publication 29 August 2025; date of current version 10 October 2025. The work of Min Chen was supported in part by the National Natural Science Foundation of China under Grant 62276109, in part by the Major Research Project of the National Social Science Foundation of China under Grant 23&ZD215. The work of Weifa Liang was supported in part by the Grants from the Research Grants Council of the Hong Kong Special Administrative Region, China CityU under Grant 11202723, CityU Grant 11202824, Grant 7005845, Grant 8730094, and Grant 9380137, respectively, in part by the National Natural Science Foundation of China under Grant 52479064, in part by the Basic and Applied Basic Research Fund of Guangdong Province under Grant 2024A1515011047 and Grant 2025A1515011022, in part by the National Natural Science Foundation of China under Grant 52479064, and in part by the Key Program of the National Natural Science Foundation of China under Grant 52539005. Recommended for acceptance by C. Ababei. (*Corresponding author: Min Chen.*)

Yu Qiu, Min Chen, and Lejun Ai are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with the Pazhou Laboratory, Guangzhou, 510006, P.R. China (e-mail: csqiyu@mail.scut.edu.cn; minchen@ieee.org).

Weifa Liang is with the Department of Computer Science, City University of Hong Kong, Hong Kong 999077, China.

Dusit Niyato is with the College of Computer Science and Engineering, Nanyang Technological University, Singapore 639956.

This article has supplementary downloadable material available at <https://doi.org/10.1109/TC.2025.3603720>, provided by the authors.

Digital Object Identifier 10.1109/TC.2025.3603720



Fig. 1. A diagram of a multi-tenant metaverse where tenants refer to three students sharing an educational metaverse. When the female interacts with the metaverse, her data is processed by the nearest server, and then the processed information is multicast to the two males.

reality IM services”, and then the metaverse would automatically allocate high-bandwidth resources and traffic routing with short paths for the specific set of users without administrator intervention.

### B. Motivation

The intent-driven information sharing process in a dynamic and resource-limited metaverse encounters the following challenges. First, unreliable services undermine user immersion. The IM services inevitably experience failures caused by hardware or software faults, disconnecting users from their tied metaverse scenario, thereby degrading their service experience. Second, the tradeoff between service reliability and infrastructure overhead is non-trivial. While adding a hot backup for an IM service can enhance its reliability, but such an arbitrary deployment increases computing resource consumption, service cost, and communication distances between the primary and backup services, as well as users in a multicast group. Finally, time-varying intents of a service provider encounter dynamic data volumes of users. User data must be processed by the IM service before multicasting, which changes its size and leads to fluctuating link usage across routing paths. Additionally, link costs, propagation delays, and available bandwidth are dynamic, while provider intentions are influenced by various factors, such as fluctuating electricity prices. These complexities hinder multicast routing to align user, network, and provider constraints. Thus, optimally placing primary and backup IM services across distinct edge servers for each sequential online multicast request while minimizing deployment overhead such as computing resources and communication distance, is a complex challenge. This difficulty is further compounded by the need to dynamically select routing trees that adapt to varying data volumes and align with dynamic natural-language-defined optimization intents.

### C. Contribution

The novelty of this paper is the first to explore intent-driven reliable information sharing in multi-tenant metaverse networks by jointly considering reliable service deployment, intent-driven multicast routing, and dynamic network

constraints including time-varying provider intents, resource capacities, infrastructure costs, and fluctuating data volumes. To address this, we propose a learning-based three-layer system to implement information sharing for a set of multicast requests arriving online.

The main contributions of this paper are as follows.

- To the best of our knowledge, we are the first to study the intent-driven reliable information sharing problem by jointly considering service deployment, multicast routing, and dynamic network constraints. We formulate the problem as a Markov decision process and prove the NP-hardness of its online version.
- To address inherent complexity of problem, we decompose it into two subproblems based on a divide-and-conquer principle: reliable service deployment and intent-driven multicast routing. We then propose a learning-based system, GTP, that integrates proximal policy optimization reinforcement learning with graph attention networks and gated recurrent units to capture dynamic graph-structured features, such as network and multicast tree information, and a Transformer encoder to evaluate correlations of each server’s attributes with user and intent constraints.
- To address reliable service deployment, we designed a deployer that places primary and backup IM services across the network by balancing computation costs and communication distances. Based on the above-calculated deployment location, for the intent-driven multicast problem, we developed a two-stage solution: a large language model-assisted intent extractor to infer time-varying provider intents, and a router that dynamically constructs multicast routing trees by jointly satisfying user demands, network constraints, and inferred intents.
- To evaluate the proposed algorithms, we conduct extensive experiments, demonstrating that GTP achieves significant improvements over state-of-the-art baselines: a 3~14% reduction in deployment overhead, 2.4~20.9% lower multicast service costs, 7~22.8% reduced multicast service latency, and up to 7% higher request throughput at most.

The rest of this article is organized as follows. Section II reviews related works. Section III introduces network, service qualities, and other mathematical models. Section IV introduces the GTP system for information sharing and mathematical analyses. Section V evaluates the performance of the proposed algorithms. Section VI concludes the paper.

## II. RELATED WORK

The metaverse is drawing increasing interest from industry and academia. Unlike prior surveys [22], [31] that focus on layered architectures from infrastructure to interaction, we categorize related work into two types based on core features: intent-driven and information-sharing. Table I highlights differences between our work and existing works.

**Intent-driven Networking:** Leivadreas et al. [11] provide a detailed analysis and summary of the five components in

TABLE I  
COMPARISON WITH RELATED WORKS

Feature/Work	[2]	[3]	[10]	[28]	[6]	[26]	[16]	[15]	[21]	[9]	Ours
Multicast	×	×	×	×	×	×	×	✓	✓	×	✓
Intent-driven	✓	✓	✓	✓	×	×	×	×	×	×	✓
Reliability	×	×	×	×	×	×	×	×	×	✓	✓
Dynamics	✓	✓	×	✓	✓	✓	✓	×	×	×	✓
Multi-tenant	×	×	×	×	×	×	×	✓	✓	×	✓
Joint optimization	×	✓	×	×	×	✓	×	×	×	×	✓

intent-driven networking for autonomous network configuration and orchestration. Following this, Collet et al. [2] proposed a loss-learning critic block to approximate flexibly complex optimization objectives designed by humans in traffic prediction, where the result of policy predictors is fed directly to the block. Then, Collet et al. [3] further designed a loss meta-learning module comprising multiple parallel individual processing units and a cascaded aggregator to predict a common goal through intertwined subpredictions. Jacobs et al. [10] introduced an intent-based system that translates natural language intents into configuration and deployment operations, such as traffic detection, and implemented it in a campus network. Wu et al. [28] designed several networking heads and a fine-tuning method for large language model (LLM) foundation models with multimodal encoding to reduce the cost and complexity in manual networking tasks, such as viewport prediction, adaptive bit-rate streaming, and cluster job scheduling.

**Information Sharing:** Du et al. [6] devised an incentive contract algorithm to improve device-to-device semantic information sharing among users in metaverse networks, aiming to reduce intensive and redundant computation in image reconstruction. Tian et al. [26] proposed a bidirectional embedding algorithm for a single interactive application to minimize consumed computation and communication resources, where multiple users with service chain and delay requirements interact with the application asynchronously in edge networks. Ma et al. [15] developed various offline and online algorithms to address single and multiple multicasting request maximization problems while minimizing computation and communication costs, assuming user requests arrive dynamically. On this basis, Ren et al. [21] reframes the aforementioned problems from the service-sharing perspective, focusing on maximizing multicast throughput at minimal cost by reusing existing services or creating new ones. Magsino et al. [16] developed a roadside unit allocation algorithm to maximize the coverage area of information sharing and vehicular connectivity for facilitating well-informed managed traffic in networks.

**Design Motivations:** Unlike existing studies that primarily focused on traffic control and adaptive loss function design for traffic forecasting in intent-driven unicasting [2], [3], [10], [28] or those limited to information routing via unicast [6], [16], [26], this paper shifts the focus to intelligent control for multicasting enabled by the natural language. Although previous studies [15], [21] explored multicasting, they overlooked the critical network dynamics, such as fluctuating data volumes before and after service processing, as well as service reliability, making their solutions insufficient for addressing the research problem in this paper. To bridge this gap, we investigate an intent-driven reliable information sharing problem in multi-tenant metaverse networks with limited resources, by

 TABLE II  
NOTATIONS

Notation	Definition
$\mathcal{G}$	The multi-tenant metaverse network
$\mathcal{V}$	The set of edge servers
$Cap_j$	The computing capacity of an edge server $v_j$
$\mathcal{E}$	The set of link $e$ connecting edge servers
$B_l, D_l, \text{ and } L_l$	The bandwidth, propagation delay, and distance of link $e_l$
$\mathcal{M}$	The set of metaverse scenarios
$m_k$	The IM service supporting $\mathcal{M}_k$ metaverse scenario
$cap_k$	The amount of computing resource demanded by each user for $m_k$
$b_k$	The amount of bandwidth needed per data unit for $m_k$
$G_k$	The multicast group including multiple users for $m_k$
$\gamma_i$	The IM service multicast request issued by user $i$
$S_i$	The edge server that is closest to the user $i$
$\mathcal{D}_i$	The set of destination servers connecting to other users in $\mathcal{M}_i$
$\mu_i$	The data volume transmitted by the user $i$
$\xi_i$	The service duration for $\mathcal{M}_i$
$Cap_{k,i}^p$	The computing resources of the primary $m_k$ for group $G_{k,i}$
$Cap_{k,i}^b$	The computing resources of the backup $m_k$ for group $G_{k,i}$
$\mathcal{T}_{k,i}^{tra}$	The transmission delay for group $G_{k,i}$ before calculating
$\mathcal{T}_{k,i}^{pro}$	The propagation delay for group $G_{k,i}$ before calculating
$\mathcal{T}_{k,i}^k$	The total communication delay from $S_i$ to server $v$ running server $m_k$
$\mathcal{T}_{k,i}^{calc}$	The calculation delay for $m_k$ to process data from user $i$ in group $G_{k,i}$
$\mathcal{T}_{k,i}^{tra}$	The transmission delay for group $G_{k,i}$ after calculating
$\mathcal{T}_{k,i}^{pro}$	The propagation delay for group $G_{k,i}$ after calculating
$\mathcal{T}_{k,i}^k$	The total communication delay from server $v$ running $m_k$ to $D_i$
$\alpha$	The correction factor used to correct transmission delay
$\beta$	The correction coefficient used to correct calculation delay
$\Delta_k$	The scaling coefficient of data volume after service $m_k$
$\mathcal{T}_i$	The total multicast delay for a single request $\gamma_i$
$c_j$	The cost per unit of computing resource for edge server $v_j$
$c_l$	The cost per unit of bandwidth for link $e_l$
$\mathcal{C}^d$	The computation costs of primary (backup) service $m_k$ issued by user $i$
$\mathcal{C}_{k,i}^{slyn}$	The synchronization cost between primary and backup services in group $G_{k,i}$
$\mathcal{C}_{k,i}^{com}$	The communication cost for sharing data across the whole multicast group $G_{k,i}$
$P_{k,i}^k$	The path connecting $S_i$ and $v$ in $\mathcal{G}$
$P_{v,D_i}^k$	The set of links consisting a tree connecting $v$ and $D_i$ in $\mathcal{G}$
$\mathcal{L}_{k,i}^p$	The synchronization distances between the primary and backup services in $G_{k,i}$
$\mathcal{L}_{k,i}^b$	The propagation distances between users in $G_{k,i}$ and the closest servers
$\mathcal{O}_i$	The service deployment overhead for request $\gamma_i$
$\epsilon_{oc}$	The weighted overhead coefficient
$ip$	The intent prompt of a service provider
$z_{ip}$	The binary variable that indicates whether the provider intent is related to delay

jointly considering service deployment, multicast routing, and dynamic network constraints, including time-varying intents, resource capacities, infrastructure costs, and fluctuating data volumes.

### III. PRELIMINARIES

In this section, we present the system models, key notions and notations listed in Table II, and the problem definitions.

#### A. Multi-Tenant Metaverse and Service Request

A multi-tenant metaverse network is represented as an undirected weighted graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V} = \{v_1, \dots, v_n\}$  is the set of edge servers, each  $v_j \in \mathcal{V}$  is equipped with computing capacity  $Cap_j$  and connects with geography-distributed base stations. Similarly,  $\mathcal{E}$  represents the set of optical links connecting edge servers, and each link  $e_l \in \mathcal{E}$  has a bandwidth capacity  $B_l$ , a propagation delay  $D_l$ , and a distance  $L_l$ . To enhance user immersion and engagement, the network provides  $|\mathcal{M}|$  types of metaverse scenarios  $\mathcal{M}_k$  supported by corresponding IM services, and for each service  $m_k$ , let  $cap_k$  be the amount of computing resource demanded by each user, and  $b_k$  the amount of bandwidth needed per data unit [15]. Multiple users requesting the same IM service  $m_k$  form a *multicast group*  $G_k$ , and multicast events in different time slots are treated as separate groups, even if they request the same IM service. Whenever a user  $i$  wishes to interact with a metaverse scenario, an IM service multicast request is generated for information sharing in  $G_{k,i} \in G_k$ , which is represented by a tuple  $\gamma_i \triangleq$



$\{\mathcal{S}_i, \mathcal{D}_i, \mathcal{M}_i, \mu_i, \xi_i\}$ , where  $\mathcal{S}_i$  is the edge server that is closest to the user who initiated the multicast session,  $\mathcal{M}_i$  specifies the metaverse scenario is enabled by the IM service  $m_{k,i}$  that users intend to join,  $\mathcal{D}_i$  represents the set of destination servers connecting to other users in  $\mathcal{M}_i$ , namely  $\mathcal{D}_i = G_{k,i} \setminus \mathcal{S}_i$ ,  $\mu_i$  represents the data volume transmitted by the user, and  $\xi_i$  is service duration. For instance, users John, Sam, and Alice actively participate in an educational metaverse for continuous information exchanges via base stations with servers in Fig. 1. In addition, let  $\mathcal{R}$  represent a set of multicast requests arrived within a time window  $T$ .

### B. Metrics of Interactive Metaverse Service

The following metrics are used to measure service qualities: reliability, delay, and cost.

**Service reliability metrics:** Service failures are inevitable, which are caused by various factors including hardware errors, software bugs, and wrong configurations. Due to an IM service being an interface to connect users with a scenario and other participants, a service failure disconnects all users from the virtual environment. To maintain the availability of a metaverse scenario in real-time, a de-facto method is to deploy a hot backup [34]. This approach refers to deploying an active backup close to the primary IM service, allowing both services to support each other. If a failure occurs, the primary one can seamlessly scale and transfer to the backup service by adopting serverless technology [12] and vice versa. The computing resources of the primary IM service  $Cap_{k,i}^p$  and backup IM service  $Cap_{k,i}^b$  required to enable participating users in a multicast group  $G_{k,i}$  are defined as follows,

$$Cap_{k,i}^p = Cap_{k,i}^b = cap_k \cdot (|\mathcal{S}_i| + |\mathcal{D}_i|), \quad (1)$$

where  $|\mathcal{S}_i|$  denotes the number of a source user, and  $|\mathcal{D}_i|$  represents the number of destination users in multicast group  $G_{k,i}$ .

**Service delay metrics:** Within a multicast group  $G_{k,i}$ , only one user has the right to multicast the message in each time slot, while the other users act as destinations, receiving the processed message. During information sharing, users must endure data calculation, transmission, and propagation delays. Specifically, the original data of user  $i$  first is transmitted to the IM service  $m_{k,i}$  hosted on an edge server  $v$  through the path  $P_{\mathcal{S}_i,v}^k$  connecting  $\mathcal{S}_i$  and  $v$  in  $\mathcal{G}$ . The communication delay  $\mathcal{T}_{\mathcal{S}_i,v}^k$  before calculating, including transmission  $\mathcal{T}_{k,i}^{tra}$  and propagation delays  $\mathcal{T}_{k,i}^{pro}$ , are defined as follows,

$$\mathcal{T}_{\mathcal{S}_i,v}^k = \mathcal{T}_{k,i}^{tra} + \mathcal{T}_{k,i}^{pro} = \alpha \cdot \mu_i + \sum_{l \in P_{\mathcal{S}_i,v}^k} D_l, \quad (2)$$

where  $\alpha$  is a correction factor used to correct transmission delay affected by different data sizes and communication protocols [21], and  $D_l$  is the propagation delay on link  $e_l$ . Next, the data is processed by the service  $m_{k,i}$  to obtain an interaction result, and the calculation delay  $\mathcal{T}_{k,i}^{cal}$  is determined by the data volume  $\mu_i$  of the user and the computing resources  $cap_k$ ,

$$\mathcal{T}_{k,i}^{cal} = \frac{\beta \cdot \mu_i}{cap_k}, \quad (3)$$

where  $\beta$  is a correction coefficient used to correct calculation delay affected by hardware and data characteristics, such as decode or decrypt [21]. Finally, the result is multicasted respectively to other participants  $d_i \in \mathcal{D}_i$  by the corresponding path  $P_{v,d_i}^k$ . The communication delay  $\mathcal{T}_{v,d_i}^k$  after calculating, including transmission  $\mathcal{T}_{k,\mathcal{D}_i}^{tra}$  and propagation  $\mathcal{T}_{k,\mathcal{D}_i}^{pro}$  delays, are

$$\mathcal{T}_{v,d_i}^k = \mathcal{T}_{k,\mathcal{D}_i}^{tra} + \mathcal{T}_{k,\mathcal{D}_i}^{pro} = \alpha \cdot \mu_i \cdot \Delta_k + \sum_{l \in P_{v,d_i}^k} D_l, \quad (4)$$

where  $\Delta_k$  is a scaling coefficient of data volume after service  $m_{k,i}$ , and  $P_{v,\mathcal{D}_i}^k = \cup_{d_i \in \mathcal{D}_i} P_{v,d_i}^k / P_{\mathcal{S}_i,v}^k$  is a union of links. Thus, the multicast delay  $\mathcal{T}_i$  for a single request  $\gamma_i$  is defined as follows<sup>1</sup>,

$$\mathcal{T}_i = \mathcal{T}_{\mathcal{S}_i,v}^k + \mathcal{T}_{k,i}^{cal} + \mathcal{T}_{v,\mathcal{D}_i}^k. \quad (5)$$

Additionally, synchronization delay is considered negligible during multicasting because the synchronization between primary and backup services can be executed at leisure.

**Service cost metrics:** The service provider charges users on a pay-as-you-go basis, so the multicast cost is a critical performance metric. The unit costs associated with edge servers and network links vary due to several factors, including geographical location, carbon emission rights, and electricity tariffs [14], [19]. Specifically,  $c_j$  denotes the cost per unit of computing resource, while  $c_l$  represents the cost per unit of bandwidth. The computation costs  $\mathcal{C}_i^d$  for the primary IM service  $\mathcal{C}_{k,i}^{cal_p}$  and its backup  $\mathcal{C}_{k,i}^{cal_b}$  are given by,

$$\mathcal{C}_{k,i}^{cal_p} = c_j \cdot Cap_{k,i}^p, \quad (6)$$

$$\mathcal{C}_{k,i}^{cal_b} = c_j \cdot Cap_{k,i}^b,$$

$$\mathcal{C}_i^d = \mathcal{C}_{k,i}^{cal_p} + \mathcal{C}_{k,i}^{cal_b}, \quad (7)$$

where the primary service and backup require different computation costs due to backup deployment location restrictions, even though they consume the same computing resources. To ensure service reliability, the primary and backup IM services need to synchronize information periodically by path  $P_{v_j,v_{j'}}^k$ , which generates the synchronization cost as follows,

$$\mathcal{C}_{k,i}^{syn} = \sum_{l \in P_{v_j,v_{j'}}^k} \mu_i \cdot b_k \cdot c_l. \quad (8)$$

In addition to the above costs, the communication cost for sharing data across the whole multicast group  $G_{k,i}$  is calculated as follows:

$$\mathcal{C}_{k,i}^{com} = \mu_i \cdot b_k \cdot \left( \sum_{l \in P_{\mathcal{S}_i,v}^k} c_l + \sum_{l' \in P_{v,\mathcal{D}_i}^k} \Delta_k \cdot c_{l'} \right). \quad (9)$$

Therefore, the multicast cost  $\mathcal{C}_i^m$  and total service cost for a single request  $\gamma_i$  is calculated as,

$$\mathcal{C}_i = \mathcal{C}_i^d + \mathcal{C}_i^m = \mathcal{C}_i^d + \mathcal{C}_{k,i}^{com} + \mathcal{C}_{k,i}^{syn}. \quad (10)$$

<sup>1</sup>The multicast delay reflects the timeliness of feedback during interactions among tenants. A shorter multicast latency enables users to perceive others and the environment more promptly, thereby enhancing the sense of immersion.



### C. Provider Intent

In real-world networks, the operational and maintenance objectives of service providers are dynamic and adaptive to various situations. For instance, when a large number of users connect to the metaverse network, the provider focuses on optimizing network performance, minimizing the service delay, and allocating more resources to meet the increasing resource demand. On the other hand, service provisioning consumes power that incurs service cost, minimizing the service cost while maximizing network throughput is the main objective of the service provider. Thus, in this paper, we concentrate on the main optimization objective: minimizing the service cost and service latency together, though proposing an intent framework. The vanilla format of the intent prompt  $ip$  is as follows<sup>2</sup>,  $z_{ip}$  is a binary variable that indicates whether the provider intent is related to delay,

$$ip = \begin{cases} \text{The objective is to reduce the service delay, } z_{ip} = 1, \\ \text{The objective is to reduce the service cost, otherwise.} \end{cases} \quad (11)$$

The “delay” and “cost” are the two keywords of the provider intent, which can be used to generate the corresponding intent prompt by a large language model, such as the ChatGPT. In addition, these intent prompts correspond to mathematical expressions in formula (5) and formula (10) for a single multicast request, respectively.

### D. Problem Definitions

The intent-driven information sharing encompasses two key subproblems: reliable IM service deployment and intent-driven multicast routing, which are defined as follows.

**Definition 1:** Given a multi-tenant metaverse network  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  and a set of users wanting to interact with a metaverse scenario  $\mathcal{M}_k$  by a multicast request  $\gamma_i$ , the reliable IM service deployment problem of a single multicast request is to place a primary and a backup IM services on two selected edge servers in  $\mathcal{G}$  so that the service deployment overhead  $\mathcal{O}_i$  is minimized, where the overhead is a weighted linear combination of computation costs  $\mathcal{C}_i^d$  of services, the synchronization distances  $\mathcal{L}_i^s$  between the primary and backup services, and the total propagation distances  $\mathcal{L}_i^d$  of data between each user in the multicast group and the closest services running on servers, subject to computing capacity on each server, namely

$$\begin{aligned} \mathcal{O}_i &= \epsilon_{oc} \cdot \mathcal{C}_i^d + (1 - \epsilon_{oc}) \cdot (\mathcal{L}_i^s + \mathcal{L}_i^d), \\ \mathcal{L}_i^s &= \sum_{l \in P_{j,j'}^k} L_l, \\ \mathcal{L}_i^d &= \sum_{l \in \min(P_{S_i,j}^k, P_{S_i,j'}^k)} L_l + \sum_{d_i \in \mathcal{D}_i} \sum_{l' \in \min(P_{j,d_i}^k, P_{j',d_i}^k)} L_{l'}, \end{aligned} \quad (12)$$

<sup>2</sup>The intent model can be easily extended to multi-class scenarios. For example, by adding a new keyword like “energy” during training (as in Section IV-B2), the model becomes a three classification and its results are shown in Fig. 8(b).

where  $\min(P_{S_i,j}^k, P_{S_i,j'}^k)$  is the smaller shortest path between path  $P_{S_i,j}^k$  from  $S_i$  to primary server  $v_j$  and path  $P_{S_i,j'}^k$  from  $S_i$  to backup server  $v_{j'}$ , and  $\epsilon_{oc}$  is the weighted overhead coefficient.

The reliable service deployment problem focuses on minimizing service deployment overhead [24], [25] to ensure that services are deployed as closely as possible to users in a multicast group while maintaining low service costs. In contrast, the following subproblem emphasizes how to share interaction information between a primary service and a source user with other destination users efficiently.

**Definition 2:** Given a multi-tenant metaverse network  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , assuming that both primary and backup IM services are deployed, a given multicast request  $\gamma_i$  requesting an IM service with the data volume and the terminal set, the intent-driven multicasting problem for the multicast request is to find a multicast tree spanning all terminal nodes to optimize dynamic intents of a service provider, namely minimization multicast delay  $\mathcal{T}_i$  or multicast cost  $\mathcal{C}_i^m$ , subject to bandwidth capacity on each link in the network.

Thus, the online intent-driven information sharing problem is defined as follows.

**Definition 3:** Given a multi-tenant metaverse network  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a dynamic provider intent, and a series of multicast requests arriving one by one sequentially where each request has personalized demands on IM services, computing resources, data volumes, and terminals (destinations), the online intent-driven information sharing problem is to first select deployment locations of primary IM services and its backup for each request, and then determine an efficient multicast tree that spans all its terminal nodes in each request, aiming to minimize dynamic provider intent within a time horizon  $\mathbb{T}$ , subject to the computing and bandwidth capacity of the network.

### E. NP-Hardness of the Defined Problems

**Theorem 1:** The reliable IM service deployment problem for a multicast request in a multi-tenant metaverse network  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is NP-hard.

*Proof:* The proof can be found in Appendix A.  $\square$

**Theorem 2:** The intent-driven multicasting problem for a multicast request in a multi-tenant metaverse network  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is NP-hard.

*Proof:* The proof can be found in Appendix B.  $\square$

## IV. INFORMATION SHARING SYSTEM

In this section, we develop an intent-driven information-sharing system GTP, combining the proximal policy optimization (PPO) reinforcement learning with feature extraction networks and multi-feature comparison networks, in a multi-tenant metaverse as depicted in Fig. 2.

To achieve this, our model leverages three core neural architectures: Graph Attention Network (GAT) [27] applies attention mechanisms to graph data, enabling adaptive weighting of neighboring nodes; Gated Recurrent Unit (GRU) [4] captures temporal dependencies in sequences through efficient gating mechanisms; and the Transformer [7] leverages self-attention to

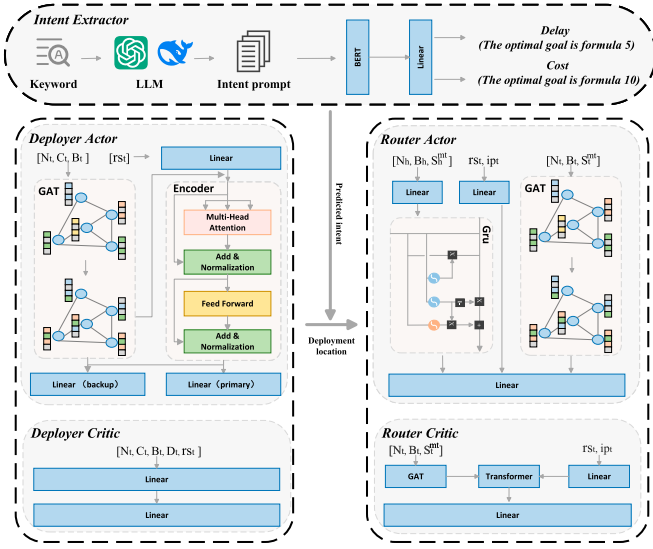


Fig. 2. The system diagram. The deployer, positioned on the left, is supported by GAT and transformer models. The extractor, located above, relies on a BERT model powered by an LLM. On the right, the intent-driven router is supported by GAT, transformer, and GRU models. In addition, specific symbols are explained in Section IV-A, where bold symbols in the figure correspond to hollow symbols in the paper.

model global dependencies in parallel across input sequences. Building on these architectures, GTP consists of three interconnected components: a reliable service deployer, an intent extractor, and an intent-driven router. Specifically, the deployer (Algorithm 1) first ensures reliability by placing backups near primary services based on network conditions and user demands. Then, the intent extractor interprets the optimization objective from natural language expressed by service providers. Finally, leveraging the outputs from both the deployer and the extractor, the intent-driven router (Algorithm 2) builds multicast trees to align information sharing with network, user, and provider constraints.

#### A. Markov Decision Process for Information Sharing

We formulate the online information sharing problem as a Markov Decision Process (MDP) with key components defined below, where the finite time horizon  $\mathbb{T}$  is divided into equal time windows  $T$ , and each consists of multiple time slots  $t$ .

**Agent:** In the multi-tenant metaverse system, two distinct agents, the deployer and the router, interact within the network environment. Each agent selects an action from the action space  $\mathbb{A}$  based on its actor network, which is driven by observations of the state space  $\mathbb{S}$ . By performing this action, the environment is transferred to a new state and the agent receives a corresponding reward  $\mathbb{R}$ .

**State space  $\mathbb{S}$ :** The state of the network environment consists of the server properties  $\mathbb{N} \cup \mathbb{C}$ , the link properties  $\mathbb{B}$ , and the multicast group properties  $rs$ . The state space is  $\mathbb{S} = \{\mathbb{N}_t, \mathbb{C}_t, \mathbb{B}_t, rs_t \mid t \in \mathbb{T}\}$ . Let  $\mathbb{N}_t = \{id_j, pos_j \mid 1 \leq j \leq |\mathcal{V}|\}$  represent the fundamental properties of edge servers. Here,  $id_j$  denotes the ID number of edge server  $v_j$ , and  $pos_j$  indicates its physical coordinates (horizontal and vertical) within the

metaverse network. The tuple  $\mathbb{C}_t = \{c_j, \tilde{Cap}_j(t), Cap_j \mid 1 \leq j \leq |\mathcal{V}|\}$  captures the computing properties of edge servers. Specifically,  $\tilde{Cap}_j(t)$  represents the available resources of the server  $v_j$  at time slot  $t$ , while  $c_j$  and  $Cap_j$  correspond to the computation cost and total capacity of the server, respectively. Similarly, the tuple  $\mathbb{B}_t = \{L_l, D_l, c_l, \tilde{B}_l(t), B_l \mid 1 \leq l \leq |\mathcal{E}|\}$  describes the link properties in the network. This includes  $L_l$ ,  $D_l$ , and  $c_l$ , which represent the distance, delay, and cost associated with link  $l$ , as well as  $\tilde{B}_l(t)$  and  $B_l$ , which denote the available and total bandwidth of link  $l$ , respectively. In addition,  $rs_t = \{id_i, pos_i, \xi_i, cap_i, \mu_i, b_i, \Delta_i\}$  represents the multicast attributes for the request  $\gamma_i$ . Here,  $id_i$  and  $pos_i$  correspond to the ID number and location of the server nearest to users in the multicast group, while  $\xi_i$ ,  $cap_i$ ,  $\mu_i$ ,  $b_i$ , and  $\Delta_i$  represent the service attributes, including the duration, computing resources, data volume, bandwidth resources, and the data scaling coefficient. Note that  $id_j$ ,  $pos_j$ , and  $L_l$  are static, while  $\tilde{Cap}_j(t)$ ,  $\tilde{B}_l(t)$ , and  $rs_t$  vary with time slot  $t$ , and the remaining properties change over the time window  $T$ . Concretely, the observed space for each agent is as follows:

- Deployment environment state  $\mathbb{S}_d$ : As the foundational module in GTP, the deployer observes the entire network state, namely

$$\mathbb{S}_d = \mathbb{S} = \{s_t^d = \{\mathbb{N}_t, \mathbb{C}_t, \mathbb{B}_t, rs_t\} \mid t \in \mathbb{T}\}. \quad (13)$$

- Route environment state  $\mathbb{S}_r$ : The Router needs to form an efficient multicast tree based on intent  $ip_t$ , thus prioritizing states of link and tree itself,

$$\mathbb{S}_r = \{s_t^r = \{\mathbb{N}_t, \mathbb{B}_t, \mathbb{S}_t^{mt}, \mathbb{S}_h^{mt}, rs_t, ip_t\} \mid t \in \mathbb{T}\}. \quad (14)$$

For request  $\gamma_i$  arriving at time slot  $t$ , its multicast tree state is  $\mathbb{S}_t^{mt} = \{s_{t,\tau}^{mt} \mid \tau < |\mathcal{V}|\}$ , where  $s_{t,\tau}^{mt} = \{\sigma_{d,j}, \sigma_{nit,j}, \sigma_{ty,j}, \sigma_{lit,j}, \sigma_{lb,j}, \sigma_{lbs,j}\}$  is tree state on the  $\tau$ -th iteration. Specifically,  $\sigma_{d,j}$  is the degree of server  $v_j$ ,  $\sigma_{nit,j}$  and  $\sigma_{lit,j} \in \{0, 1\}$  represent whether the server and link are part of the multicast tree,  $\sigma_{ty,j} \in \{1, 0, -1, -2\}$  indicates the node type (source, others, service node, and destination), as well as  $\sigma_{lb,j}$  and  $\sigma_{lbs,j} \in \{0, 1\}$  denote whether the available bandwidth of link  $e_l$  exceeds the original bandwidth required by multicast and scaling bandwidth after IM service processing, respectively. In addition,  $\mathbb{S}_h^{mt} = \{s_{t,\tau-h}^{mt}, \dots, s_{t,\tau}^{mt}\}$  is the history state of multicast tree, where any state  $s_{t,\tau-h}^{mt}$  with  $\tau - h < 0$  is replaced by  $s_{t,0}^{mt}$ .

**Action space  $\mathbb{A}$ :** Based on the observed state  $s_t = \{s_t^d, s_t^r\}$  at slot  $t$ , the agent selects and executes the action with the highest value from the action space  $\mathbb{A}$ , namely,

- Deployment action  $\mathbb{A}_d$ : The deployment action defines the placement locations of IM services required by the request  $\gamma_i$  on the network at slot  $t$ ,

$$\mathbb{A}_d = \{a_t^d = \{a_{p,t}^d, a_{b,t}^d\} \mid t \in \mathbb{T}\}, \quad (15)$$

where  $a_{p,t}^d = \arg \max \{x_{i,j} \mid 1 \leq j \leq |\mathcal{V}|\}$  is the location of the primary IM service, and  $x_{i,j}$  represents the logistic values for placing the primary service  $m_i$  on servers  $v_j$ .  $a_{b,t}^d = \arg \max \{\tilde{x}_{i,j} \mid 1 \leq j \leq |\mathcal{V}|\}$  is location of the backup IM

service, and  $\tilde{x}_{i,j}$  indicates the logistic values for placing the backup service  $m_i$  on servers  $v_j$ .

- Route action  $\mathbb{A}_r$ : The route action is to find an efficient multicast tree in  $\mathcal{G}$  to accommodate multicast request  $\gamma_i$  at slot  $t$ , given by

$$\mathbb{A}_r = \{a_t^r = \{a_{t,\tau}^r \mid 1 \leq \tau < |\mathcal{V}|, t \in \mathbb{T}\}, \quad (16)$$

where  $a_{t,\tau}^r = \arg \max \{y_{\tau,j} \mid 1 \leq j \leq |\mathcal{V}|\}$  represents the next node to be added to the Steiner tree at  $\tau$ -th iteration of the tree construction process, and  $y_{\tau,j}$  denotes the logistic value for selecting server  $v_j$ .

**State transition:** A state transition occurs after the agent executes the selected actions. Due to the hierarchical relationship between deployment and routing actions, the state transition process follows a similar structure. For each multicast request, the deployment action occurs first, keeping the network state stable at  $s_t$ . Then, multiple rounds of routing actions are performed, each selecting a new node for the multicast tree, transitioning the state from  $s_{t,\tau}^{mt}$  to  $s_{t,\tau+1}^{mt}$ . Once all the above actions are performed, the state transitions from  $s_t$  to  $s_{t+1}$ , following the conditional probability  $p(s_{t+1}|s_t, a_t)$  that is unknown to the agent.

**Reward function  $\mathbb{R}$ :** Reward acts as a feedback signal to assess action in achieving final goal from the current state. Reward function of each agent is different, as given below.

- Deployment reward  $\mathbb{R}_d$ : The goal of the deployer is to minimize the overheads for consecutively arriving multicast requests, so the reward function  $r_t^d$  at slot  $t$  is as follows,

$$r_t^d = \begin{cases} \max(\epsilon_{rd} - \mathcal{O}_i, 0), & \text{if deploy success,} \\ \mathcal{P}_i^d, & \text{otherwise,} \end{cases} \quad (17)$$

where  $\mathcal{P}_i^d$  is the penalty incurred when server capacity is insufficient to accommodate the services,

$$\mathcal{P}_i^d = -\epsilon_{pd} \cdot (\max(\text{Cap}_{k,i}^p - \tilde{\text{Cap}}_j(t), 0) + \max(\text{Cap}_{k,i}^b - \tilde{\text{Cap}}_j'(t), 0)), \quad (18)$$

$\epsilon_{rd}$  and  $\epsilon_{pd}$  are the constant values for deployment reward and penalty, respectively.

- Route reward  $\mathbb{R}_r$ : For one-by-one arriving multicast requests, the goal of the router for each request is to construct an efficient multicast tree by iteratively selecting a node and a corresponding edge that meets constraints with the lowest link attributes (i.e.,  $c_l$  and  $D_l$ ) to add to the partial constructed tree, and this procedure is similar to the Prim algorithm for minimum spanning trees. Let  $\mathcal{I}_{i,\tau}$  denote the link attribute value for request  $\gamma_i$  on the  $\tau$ -th iteration, where

$$\mathcal{I}_{i,\tau} = (1 - z_{ip}) \cdot c_l \cdot 100 + z_{ip} \cdot D_l. \quad (19)$$

To prioritize consider destination nodes  $d_i \in \mathcal{D}_i$  during tree construction, a link legality function  $\mathcal{F}(\tau)$  is introduced, where  $\epsilon_{r_r}$  and  $\epsilon_{p_r}$  are constants for routing reward

and penalty,

$$\mathcal{F}(\tau) = \begin{cases} \epsilon_{r_r}, & \exists d_i \text{ is selected,} \\ -\epsilon_{p_r}, & \forall d_i \text{ is not optional,} \\ -2 * \epsilon_{p_r}, & \exists d_i \text{ is optional but unselected,} \end{cases} \quad (20)$$

Thus, the reward function of tree construction at  $\tau$ -th iteration at slot  $t$  is

$$r_{t,\tau}^r = \begin{cases} \max(\epsilon_{r_r} - \mathcal{I}_{i,\tau}, 0) + \mathcal{F}(\tau), & \text{if link success,} \\ \mathcal{P}_{i,\tau}^r, & \text{otherwise,} \end{cases} \quad (21)$$

and  $\mathcal{P}_{i,\tau}^r$  is a penalty for insufficient bandwidth, namely the dynamic data volume  $\tilde{b}_i$  is larger than the link bandwidth, and even the request is rejected,

$$\mathcal{P}_{i,\tau}^r = -\epsilon_{pb} - \epsilon_{p_r} \cdot \max(\tilde{b}_i - \tilde{B}_l(t), 0), \quad (22)$$

$$\tilde{b}_i = \mu_i \cdot b_i \cdot (\tilde{z}_\tau + (1 - \tilde{z}_\tau) \cdot \Delta_i), \quad (23)$$

where  $\tilde{z}_\tau$  is a binary variable indicating whether the link in  $\tau$ -th iteration transmits scaled data, and  $\epsilon_{pb}$  is also a constant for routing penalty.

## B. Learning-Based Algorithms for Information Sharing

The rest is to develop a learning-based algorithm for the service deployer, followed by a fine-tuning algorithm for the intent extractor, and conclude with an intent-driven router. Note that each time window represents a training episode.

1) *The Service Deployer:* The service deployer as a foundational module for information sharing, plays a crucial role in two key areas: 1) it ensures a reliable and immersive metaverse experience by deploying additional hot backups close to primary IM services; and 2) it optimizes implicitly dynamic provider intent by carefully considering both the service computation cost and communication distance. The detail of Algorithm 1 is given as follows.

First, the deployment agent observes network states  $\mathbb{N}_t$ ,  $\mathbb{C}_t$ , and  $\mathbb{B}_t$  at slot  $t$ , and inputs it into the GAT to get network embedding  $\mathbf{e}_n$  through a graph-structured feature extraction process. Specifically, 1) attention coefficients  $\hat{\alpha}_{j,j'}$  are derived from a shared linear transformation with trainable weight matrix  $\mathbb{W} \in R^{F' \times F}$  and a trainable weight vector  $ATT \in R^{2F'}$ , where  $\hat{\alpha}_{j,j'} = ATT(\mathbb{W}\vec{f}_j, \mathbb{W}\vec{f}_{j'})$ , which represent the significance of eigenvector  $\vec{f}_j = \{\mathbb{N}_{t,j}, \mathbb{C}_{t,j}, \mathbb{B}_{t,j}\}$  of edge server  $v_j$  to that of server  $v_{j'}$ . Then, 2) the softmax function is applied to normalize the attention coefficients  $\hat{\alpha}_{j,j'}$  and obtain the attention probabilities  $\bar{\alpha}_{j,j'} = \frac{\exp(\hat{\alpha}_{j,j'})}{\sum_{j' \in \mathcal{N}_j} \exp(\hat{\alpha}_{j,j'})}$ , where only the neighbor nodes  $\mathcal{N}_j$  of server  $v_j$  are involved in the calculation. After that, 3) the new eigenvector  $\vec{f}_j^l$  of server  $v_j$  is obtained by considering the information of its neighbors, where  $\vec{f}_j^l = \zeta(\sum_{j' \in \mathcal{N}_j} \bar{\alpha}_{j,j'} \cdot \mathbb{W}\vec{f}_{j'})$  and  $\zeta(\cdot)$  is an activation function. Note that when multi-head attention mechanisms are used, the average function is applied, namely  $\vec{f}_j^l = \zeta(\frac{1}{K} \sum_{k=1}^K \sum_{j' \in \mathcal{N}_j} \bar{\alpha}_{j,j'}^k \cdot \mathbb{W}^k \vec{f}_{j'})$ , where  $\bar{\alpha}_{j,j'}^k$  and  $\mathbb{W}^k \vec{f}_{j'}$  is the  $k$ -th weight network. Thus, the process of



**Algorithm 1: The Train of Reliable Deployer**


---

**Input:** A multi-tenant metaverse network  $\mathcal{G}$ , and a set  $\mathcal{R}$  of multicast requests arrive one by one.  
**Output:** A IM service deployment schemes.

---

```

1 begin
2   for each episode  $ep \in EP$  do
3     for each multicast request  $\gamma_i \in \mathcal{R}$  do
4       Deployer agent observes deployment state  $s_t^d$ , consisting a
5       network state  $\mathbb{N}_t$ ,  $\mathbb{C}_t$ ,  $\mathbb{B}_t$ , and a request state  $rs_t$  at slot  $t$ ;
6       Generate a network and a user embedding by using a GAT
7       and an FC network to process related states, respectively;
8       Using a Transformer to calculate the self-attention of these
9       embeddings, deployer selects a primary  $dl_i^p$  and backup  $dl_i^b$ 
10      deployment to host required IM services, and obtains a
11      reward  $r_t^d$ ;
12      if Deployment locations do not exist then
13        Reject the request  $\gamma_i$ , continue;
14      else
15        Transaction  $(s_t^d, \{dl_i^p, dl_i^b\}, s_{t+1}^d, r_t^d)$  is stored in
16        deploy replay buffer;
17    if  $ep \% \overline{ep}_d == 0$  then
18      Extract a mini-batch  $\mathcal{B}_d$  of data from the buffer, and neural network
19      of deployer is updated multiple times by gradient descent through
20      formula (26) and (27);

```

---

obtaining converged eigenvector  $\vec{f}_j'$  of each server by  $\vec{f}_j$  through the  $q$ -th layer of GAT is expressed as

$$\vec{f}_j' = GAT^q(\vec{f}_j, \vec{f}_{j'}), \forall j' \in \mathcal{N}_j, \quad (24)$$

Next, the multicast group state  $rs_t$  of request  $\gamma_i$  at slot  $t$  is passed through a fully-connected (FC) neural network to obtain the user embedding  $\epsilon_u$ . The combined embedding  $\epsilon_c \in \mathbb{R}^{(|\mathcal{V}|+1) \times F'}$ , including  $\epsilon_n$  and  $\epsilon_u$ , is then passed to the Transformer encoder to evaluate correlations of each server's abilities with user constraints. For each element in combined embedding  $\epsilon_c$ , 1) a learned weight matrix maps it to Query  $\Omega = \epsilon_c \mathcal{W}_q$ , Key  $\mathcal{K} = \epsilon_c \mathcal{W}_k$ , and Value  $\mathcal{V} = \epsilon_c \mathcal{W}_v$ . Then, 2) the correlation  $\tilde{\alpha} = \text{softmax}\left(\frac{\Omega \mathcal{K}^T}{\sqrt{d_k}}\right) \mathcal{V}$  is calculated by the self-attention mechanism. Note that when using multi-head attention, each head has its own set of query, key, and value matrices, and the attention outputs are concatenated and passed through a final output matrix  $\mathcal{W}_o$ . After that, 3) the resulting output  $\tilde{\epsilon}_c$  is processed through a feedforward neural network, followed by residual connections and layer normalization, producing the final output  $\hat{\epsilon}_c$  of the combined embedding. Thus, the process of obtaining the adjusted combined embedding  $\hat{\epsilon}_c$  after the  $q$ -th layer of the Transformer encoder is expressed as

$$\hat{\epsilon}_c = \text{TRA}^q(\epsilon_c). \quad (25)$$

Finally, the  $\hat{\epsilon}_c$  is fed into both the primary and backup service placement policies to determine the placement locations ( $dl_i^p$  and  $dl_i^b$ ) of the primary and backup services, respectively. To ensure mutual exclusion of deployment locations, a masking mechanism is introduced, where the primary deployment location, once selected, is masked when choosing the backup. Additionally, to preserve the relevance of the deployment relationship, the graph eigenvector  $\vec{f}_j'$  of the selected primary server  $v_j$  is used when selecting the backup location. Once the deployment locations are successfully selected, the relevant

transactions  $(s_t^d, \{dl_i^p, dl_i^b\}, s_{t+1}^d, r_t^d)$  are stored in the memory pool. When the number of transactions in the memory pool reaches a predefined threshold, the critic  $\mathcal{V}_\phi(s_t^d)$  and actor  $\mathcal{A}_\theta(s_t^d, a_t^d)$  network is updated by gradient descent for formulas (26) and (27) [23],

$$\mathcal{L}_d(\phi) = \mathbb{E} \left[ \mathcal{V}_\phi(s_t^d) - \sum_{j=t}^{|\mathcal{B}_d|} \omega^{j-t} r_t^d \right], \quad (26)$$

$$\mathcal{L}_d(\theta) = \mathbb{E}[\min(\lambda(\theta), \text{clip}(\lambda(\theta))) \cdot \mathcal{A}_{\theta_{old}}(s_t^d, a_t^d)], \quad (27)$$

where  $\lambda(\theta) = \frac{\pi_\theta(a_t^d | s_t^d)}{\pi_{\theta_{old}}(a_t^d | s_t^d)}$ ,  $\mathcal{A}_{\theta_{old}}(s_t^d, a_t^d) = \delta_t + (\varphi\omega) \cdot \delta_{t+1} + \dots + (\varphi\omega)^{|\mathcal{B}_d|-1} \cdot \delta_{|\mathcal{B}_d|}$ ,  $\delta_t = r_t^d + \omega \mathcal{V}_\phi(s_{t+1}^d) - \mathcal{V}_\phi(s_t^d)$ , and  $\text{clip}(\cdot)$  is a truncation function that limits the value to a given range  $[1 - \epsilon, 1 + \epsilon]$ . In addition,  $\omega$  and  $\varphi \in [0, 1]$  are attenuation coefficients.

2) *The Intent Extractor*: The intent extractor as one module for information sharing, is responsible for identifying the core intentions of service providers expressed in natural language, facilitating more automated management and maintenance of the metaverse network. The pseudo-code of the extractor can be found in Appendix C, and the details are described as follows.

First, intent sentences are created based on the core keywords specified by the service provider. For instance, for “delay” keyword, we enter it into the ChatGPT with the prompt “Please provide  $m$  sentences describing the service provider’s optimization requirements for service latency from a first-person perspective”. Then, the index of each keyword in all intent types is as the label, and the generated intent sentences are as data to form a training database. Finally, we train a BERT-based intent classifier using the above dataset and the mean square error loss function. Compared to traditional approaches, the use of an LLM-enabled extractor offers two key advantages. First, it enables the collection of diverse ranges of intent expressions related to the optimization objectives, thereby enhancing the generalization ability of the trained model. Second, it significantly reduces the cost associated with intent data collection.

3) *The Intent-Driven Router*: The intent-driven router as one important module of information sharing, combines the outputs from the service deployer with the intent extractor to build multicast (Steiner) trees to meet the requirements of networks, users, and the service provider intents. Later, Steiner trees are referred to as multicast trees, and they are interchangeable. The details of Algorithm 2 for the information sharing problem are given as follows.

First, a random intent prompt is generated, and a Steiner tree for request  $\gamma_i$  arriving at slot  $t$  is initialized. Specifically, we construct a new Steiner tree  $mt_{t,0} = \{dl_i^p\}$  for request  $\gamma_i$ , with  $dl_i^p$  as the root, and update the destinations to  $\hat{\mathcal{D}}_i = \mathcal{S}_i \cup dl_i^b \cup \mathcal{D}_i \setminus dl_i^p$ , this is because the volume of data changes when processed through the server hosting the IM service. We prioritize the source user  $\mathcal{S}_i$  and the backup location  $dl_i^b$ , which optimizes the multicast tree construction, by first selecting the link for transmitting the raw data, followed by choosing the link for transmitting the scaled data.

Then, the route agent observer the network state ( $\mathbb{N}_t$  and  $\mathbb{B}_t$ ), tree state ( $\mathbb{S}_{t,\tau}^{mt}$  and  $\mathbb{S}_h^{mt}$ ), request state  $rs_t$ , and provider

**Algorithm 2: The Train of Intent-driven Router**


---

**Input:** A multi-tenant metaverse network  $\mathcal{G}$ , and a set  $\mathcal{R}$  of multicast requests arrive one by one, and IM location  $dl_i^p$  and  $dl_i^b$  for each request.

**Output:** A IM service multicast routing schemes.

```

1 begin
2   for each episode  $ep \in EP$  do
3     Generate a random intent prompt  $ip_t$ ;
4     Put  $ip_t$  into the BERT-based intent extractor to obtain
       corresponding result  $\hat{ip}_t$ ;
5     for each multicast request  $\gamma_i \in \mathcal{R}$  do
6       Form a new Steiner tree  $mt_{t,0} = \{dl_i^p\}$  for request  $\gamma_i$ , and
       a set of modified destinations  $\hat{\mathcal{D}}_i = \mathcal{S}_i \cup dl_i^b \cup \mathcal{D}_i \setminus dl_i^p$ ;
7       while Destination set  $\hat{\mathcal{D}}_i$  is not empty do
8         Router agent observes route state  $s_t^r$  at slot  $t$ , consisting
            $\mathbb{N}_t, \mathbb{B}_t, \mathbb{S}_t^{mt}, \mathbb{S}_t^{nt}, r_{st}$ , and  $\hat{ip}_t$  on the  $\tau$ -th iteration;
9         The  $s_t^r$  is fed into GAT, GRU, and FC networks to
           generate the corresponding embeddings, respectively;
10        Based on the above embeddings and neighbor mask
            $nm_\tau$ , router first selects a target node  $tn$  and then
           select greedily a source node  $sn$  in the tree  $mt_{t,\tau}$  to
           form a new link  $e_{sn,tn}$ ;
11        if  $sn$  does not exist then
12          Reject the request  $\gamma_i$ , break;
13        Integrate  $e_{sn,tn}$  into the tree  $mt_{t,\tau}$ , transitioning to a
           new tree state  $s_{t,\tau+1}^{mt}$ , and obtaining reward  $r_{t,\tau}^r$ ;
14        if  $tn$  in destination set  $\hat{\mathcal{D}}_i$  then
15          Delete the node  $tn, \hat{\mathcal{D}}_i \leftarrow \hat{\mathcal{D}}_i \setminus tn$ ;
16        Transaction  $(s_{t,\tau}^r, e_{sn,tn}, r_{t,\tau}^r, s_{t,\tau+1}^r)$  is stored in
           route replay buffer;
17      if  $ep \% \bar{ep}_r == 0$  then
18        Extract a mini-batch  $\mathcal{B}_r$  of data from the buffer, and neural
           network of router is updated multiple times by gradient
           descent through formula (28) and (29);

```

---

intent  $ip_t$  to select the links needed to construct the multicast tree. Specifically, the concatenation of  $\mathbb{N}_t, \mathbb{B}_t$ , and  $\mathbb{S}_{h,t,\tau}^{mt}$  is fed into GAT network to generate tree graph embedding  $\epsilon_g$ , while tree history states  $\mathbb{S}_h^{mt}$  is fed to gated recurrent unit network to get tree time-series embedding  $\epsilon_{ts}$ . To align with tree-related embeddings,  $\hat{ip}_t$  and  $\gamma_i$  are processed by the FC network to generate corresponding user embedding  $\epsilon_u$ , and intent embedding  $\epsilon_i$ . The combined embedding  $\epsilon_{ct} = [\epsilon_{ts} + \epsilon_u + \epsilon_i, \epsilon_g]$  is then input into the link selection policy to determine the next node  $tn$ , which forms an endpoint of the new link  $e_{sn,tn}$ , while a greedy approach selects the source node  $sn$  in the tree  $mt_{t,\tau}$  based on the attribute values of  $e_{sn,tn}$ . Note that we use the neighbor mask  $sn$  during selecting  $tn$ , which focuses solely on the one-hop servers with the merged servers in the tree.

Finally, after successfully selecting link  $e_{sn,tn}$ , the corresponding transactions  $(s_{t,\tau}^r, e_{sn,tn}, r_{t,\tau}^r, s_{t,\tau+1}^r)$  are added to the memory pool. When the number of transactions in the memory pool surpasses a pre-defined threshold, the critic  $\mathcal{V}_\phi(s_{t,\tau}^r)$  and actor  $\mathcal{A}_\theta(s_{t,\tau}^r, a_{t,\tau}^r)$  network is updated by gradient descent for formulas (28) and (29) [23].

$$\mathcal{L}_r(\hat{\phi}) = \mathbb{E} \left[ \mathcal{V}_\phi(s_{t,\tau}^r) - \sum_{j=t}^{|\mathcal{B}_r|} \omega^{j-t} r_t^r \right], \quad (28)$$

$$\mathcal{L}_r(\hat{\theta}) = \mathbb{E}[\min(\lambda(\hat{\theta}), \text{clip}(\lambda(\hat{\theta}))) \cdot \mathcal{A}_{\hat{\theta}_{old}}(s_{t,\tau}^r, a_{t,\tau}^r)], \quad (29)$$

where  $\lambda(\hat{\theta}) = \frac{\pi_{\hat{\theta}}(a_{t,\tau}^r | s_{t,\tau}^r)}{\pi_{\hat{\theta}_{old}}(a_{t,\tau}^r | s_{t,\tau}^r)}$ ,  $\mathcal{A}_{\hat{\theta}_{old}}(s_{t,\tau}^r, a_{t,\tau}^r) = \delta_t + (\varphi\omega) \cdot \delta_{t+1} + \dots + (\varphi\omega)^{|\mathcal{B}_r|-1} \cdot \delta_{|\mathcal{B}_r|}$ , and  $\delta_t = r_{t,\tau}^r +$

**Algorithm 3: The Intent-driven information sharing system, named GTP**


---

**Input:** A dynamic metaverse network  $\mathcal{G}$ , a set  $\mathcal{R}$  of multicast requests arrive one by one, and well-trained deployer, extractor, and router.

**Output:** Information sharing schemes.

```

1 begin
2   for each time window  $T \in \mathbb{T}$  do
3     Generate a random intent prompt  $ip_t$ ;
4     Reset the network capacity, infrastructure price, and user multicast
       demands in the multi-tenant metaverse;
5     for An arrival multicast request  $\gamma_i \in \mathcal{R}$  at  $t$  do
6       Observe deployment state  $s_t^d$  and select primary  $dl_i^p$  and
           backup  $dl_i^b$  deployment locations by the reliable service
           deployer;
7       Put  $ip_t$  into the intent extractor to obtain corresponding result
            $\hat{ip}_t$  by the intent extractor;
8       Observe route state  $s_t^r$  and generate a Steiner tree by the
           intent-driven router;
9       if Deploy and route are successful then
10        Receive  $\gamma_i$  and allocate resources;

```

---

$\omega \mathcal{V}_\phi(s_{t,\tau+1}^r) - \mathcal{V}_\phi(s_{t,\tau}^r)$ . Note that the critic network uses GAT and transformed network when inference, as shown in Fig. 2, whose principle is similar to the deployer in Section IV-B1.

4) *The Intent-Driven Information Sharing System:* Once the service deployer, intent extractor, and intent-driven router models are trained as described above, real-time information sharing can be enabled in the dynamic, multi-tenant metaverse. The detail of Algorithm 3 is given as follows. Recall that the finite time horizon  $\mathbb{T}$  is partitioned into equal time windows  $T$ , with each window further divided into multiple time slots  $t$ .

First, at the beginning of each time window  $T$ , we randomly generate an intent prompt from the service provider and simulate the dynamic evolution of the multi-tenant metaverse simultaneously. Namely, we reset the properties of the underlying network including computing capacity  $Cap_j$ , bandwidth capacity  $B_l$ , infrastructure costs  $c_j$  and  $c_l$ , link delay  $D_l$ , and multicast information in the request set  $\mathcal{R}$ . Note that the arrival time of each request is different within the time window  $T$ . For an incoming multicast request  $\gamma_i$  at slot  $t \in T$ , we determine the deployment location of primary and backup IM services by invoking the deployer trained by Algorithm 1, followed by identifying the server provider intent by invoking the intent extractor, and we finally find a specific multicast routing tree by invoking the intent-driven router trained by Algorithm 2. If the procedure goes smoothly, the relevant resources are allocated and reliable information sharing begins; otherwise the request is denied. This procedure continues within the time window  $T$  until all requests in group  $\mathcal{R}$  have been processed.

### C. Algorithm Analysis

*Theorem 3:* Given a metaverse network  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a dynamic provider intent, and a set  $\mathcal{R}$  of multicast requests with demands on IM services, computing resources, data volumes, and terminals, where each server in  $\mathcal{G}$  has computing capacity  $Cap_j$  with computation cost  $c_j$ , each link in  $\mathcal{G}$  has bandwidth capacity  $B_l$  with bandwidth cost  $c_l$ , the propagation delay  $D_l$ , there is an efficient learning-based system GTP, and there is

an algorithm Algorithm 3 built upon the system, which takes  $\mathcal{O}(|\mathcal{R}||\mathcal{V}||\mathcal{E}|)$  time complexity per time window  $T$  to implement the intent-driven reliable information sharing in dynamic metaverse.

*Proof:* The proof can be found in Appendix D.  $\square$

## V. PERFORMANCE EVALUATION

In this section, we present the experimental parameter settings first, followed by introducing several baseline algorithms. Finally, we analyze the impact of key parameters on the performance of the algorithms in a simulated multi-tenant metaverse network.

### A. Experimental Settings

We simulated a multi-tenant metaverse network, using the Abilene network topology [19], covering an area of 5,500 km by 5,500 km, with 11 operational edge servers. Since the location  $pos_j$  of each server  $j \in \mathcal{V}$  is given, the link distance  $L_l$  between two servers is known in advance, and the computing capacity  $Cap_j$  of server  $j$  is randomly drawn in the range [1,000, 3,000] *MHz* with the cost of  $c_j$  per unit computing resource and the value of  $c_j$  is drawn in the range of [1, 10]\$ [20]. The bandwidth capacity  $B_l$  of each link  $l \in \mathcal{E}$  is randomly chosen from [1,000, 3,000] *Mbps*, and the link delay  $D_l$  and transmission cost  $c_l$  are randomly drawn from [10, 100] *ms* and [0.1, 1]\$, respectively [32]. To provide a diverse metaverse experience, the network offers 5 types of IM services  $m_k$  with  $1 \leq k \leq 5$ . Each service requires the amount of computing resources  $cap_k$  in [100, 200] *MHz* per user and the amount of bandwidth resources  $b_k$  [1, 5] *Mbps* per data unit. The scaling coefficient  $\Delta_k$  for the data volume after service  $m_k$  is randomly selected in the range of [0.1, 0.9].

When a user wants to interact with a desired metaverse scenario, a multicast request  $\gamma_i$  is issued, where the scenario is randomly selected from 5 types, with the required data volume  $\mu_i$  ranging from [50, 100] *Mb* and the service duration  $\xi_i$  randomly set between 20 and 50 slots. In a metaverse scenario at a given time slot, a multicast group consists of 5 users, with their horizontal and vertical coordinates  $pos_i$  is randomly selected from [0, 5000] km. The servers closest to each user in the multicast group are designated as the source  $\mathcal{S}_i$  and destination  $\mathcal{D}_i$  servers, with the server nearest to the issuing user serving as the source. Requests follow a Poisson process with an average arrival rate  $k_p$  of 0.025 [5], resulting in 100 requests within a time window  $T = 400$  slots. Additionally, the following coefficients are set: the transmission delay correction factor  $\alpha$ , and the calculation delay correction factor  $\beta$ , are both set to 1; the weighted overhead coefficient  $\epsilon_{oc}$  is 0.7; the deployment reward constant  $\epsilon_{r_d}$  is 150, while the deployment penalty constant  $\epsilon_{p_d}$  is 50; the routing reward constant  $\epsilon_{r_r}$  is 100, the routing penalty constant  $\epsilon_{p_r}$  is 100, and the bandwidth penalty constant  $\epsilon_{p_b}$  is 200.

The structure and parameters of the designed neural network are given as follows, also as shown in Fig. 2. The dimension of the output layer of the FC network is  $|\mathcal{V}|$  for actor networks, otherwise 128. In addition, for the deployer, the actor network

TABLE III  
PARAMETER SETTINGS

	$Cap_j$	$c_j$	$\epsilon_{oc}$	$ip$	$c_l$	$D_l$	$B_l$	$\Delta_k$
Set1	[1,000–3,000]	1–10	0.7	0 or 1	0.1–1	10–100	1,000–3,000	0.1–1
Set2	1,000–3,000	[1.5–9.5]	0.7	0 or 1	0.1–1	10–100	1,000–3,000	0.1–1
Set3	1,000–3,000	1–10	0.3, 0.5, 0.7	0 or 1	0.1–1	10–100	1,000–3,000	0.1–1
Set4	1,000–3,000	1–10	0.7	from 0 to 1	0.1–1	10–100	1,000–3,000	0.1–1
Set5	1,000–3,000	1–10	0.7	0 or 1	[0.15–0.95]	10–100	1,000–3,000	0.1–1
Set6	1,000–3,000	1–10	0.7	0 or 1	0.1–1	[15–95]	1,000–3,000	0.1–1
Set7	1,000–3,000	1–10	0.7	0 or 1	0.1–1	10–100	[1,100–2,700]	0.1–1
Set8	1,000–3,000	1–10	0.7	0 or 1	0.1–1	10–100	1,000–3,000	[0.1–0.9]

utilizes both a Transformer and a GAT network to extract features, followed by two three-layer FC network to select respectively the backup service deployment location based on these features, where both the Transformer and GAT networks consist of 8 layers, with each layer having 16 head nodes. The critic network uses a five-layer FC network to evaluate the value of an action. For the intent extractor, BERT network consists of a 12-layer Transformer encoder, each layer having 768 hidden units and 12 attention heads, with a total of around 110 million parameters. For the router, the actor network combines a GAT and a GRU network to extract features, followed by a two-layer FC network to sequentially select the next node in a new link. Both the GAT with 16 head nodes and GRU networks consist of 3 layers. The critic network uses a three-layer Transformer and FC network to evaluate the value of action. In addition, the batch sizes for the deployer, BERT, and router are set to 1000, 128, and 4096, respectively, while their corresponding learning rates are  $3 \times 10^{-4}$ ,  $2 \times 10^{-5}$ , and  $3 \times 10^{-4}$ .

To evaluate the performance, we consider the following algorithms, with detailed experimental configurations given in Table III, where  $\{\cdot - \cdot\}$  indicates that the range is divided into some sub-intervals, and each  $k$ -th experiment value is randomly selected from within the  $k$ -th sub-interval.

- **OPT:** This algorithm assumes the objective functions are given in advance, which always selects the optimal actions by exhaustive search. Its time complexity is prohibitively high due to the exhaustive search on the solution space.
- **GTP:** This algorithm, proposed in this paper, combines PPO architecture with GAT and Transformer networks to provide an efficient solution.
- **DSAC\_GT:** This algorithm combines the Deep Soft Actor-Critic (DSAC) architecture with GAT and Transformer networks to provide an efficient solution. Unlike GTP, which is an online reinforcement learning algorithm, DSAC\_GT is an offline reinforcement learning algorithm. Next, for the reliable IM service deployment problem, three comparison algorithms are introduced too.
- **PPO\_GrT [1]:** This algorithm combines PPO architecture with GRU and Transformer networks to select deployment locations for primary and backup IM services, leveraging the current state as well as a sequence of historical states.
- **D3QN\_T [29]:** This algorithm integrates the Dueling Double Deep Q-Network (D3QN) architecture with Transformer network to determine deployment locations, lacking of the feature extraction from additional neural networks.
- **SA [8]:** The algorithm is a heuristic-based approach that primarily employs simulated annealing techniques to determine the deployment location.



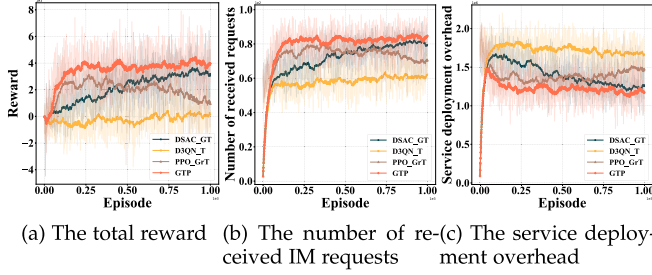


Fig. 3. The performance of deployers during model training.

Finally, for the intent-based multicast routing problem, three more comparison algorithms are presented.

- **D3QN\_Gr [13]:** This algorithm combines the D3QN architecture with a GRU network to select the links that form the multicast tree, utilizing both the current state and a series of historical tree states.
- **DQN\_Gc [33]:** This algorithm integrates the DQN (Deep Q-Network) architecture with a GCN (Graph Convolutional Network) to select links for constructing the multicast tree
- **DQN [30]:** The algorithm uses only DQN network select links to build the multicast tree without any feature extraction.

### B. Performance Evaluation of the Proposed System

The following is dedicated to evaluating the performance of the proposed algorithms for the system.

1) *Impact of Key Parameters on Different Deployers:* We trained all service deployers with a random seed of 123 over  $1 \times 10^3$  training steps initially. The detailed evaluation results are as follows.

**Training process:** For different deployers, Figs. 3 and 4 illustrate the trend on total rewards, number of accepted requests, and service deployment overhead during model training and testing, respectively. As shown in Fig. 3(a), as the number of training steps increases, the reward of all algorithms first rises and then stabilizes, except for the PPO\_GrT algorithm. Among the methods, GTP consistently achieves the highest reward and converges most quickly, followed by DSAC\_GT. Fig. 3(b) and (c) show the reasons behind this, despite the increase in the number of accepted requests, the service deployment overhead continues to decrease for most algorithms, except for the PPO\_GrT algorithm, which accepts fewer requests. The same trend is observed in Fig. 4, where the overhead and received request of the GTP algorithm is second only to the optimal OPT algorithm, while PPO\_GrT and D3QN\_T show poor performance due to their lack of feature extraction capabilities. Note that SA and OPT are non-machine learning-based methods and therefore do not require the training phase.

Then, we investigated the impact of key parameters, such as service capacity, server cost, and the overhead coefficient, on the key performances of the various algorithms.

**Service capacity:** Fig. 5 shows the impact of the service capacity  $Cap_j$  on the proposed deployers by varying its average

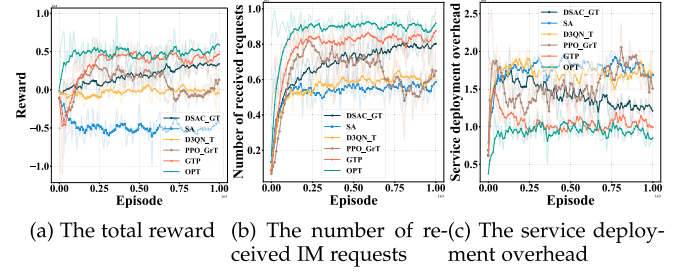


Fig. 4. The performance of deployers during model test.

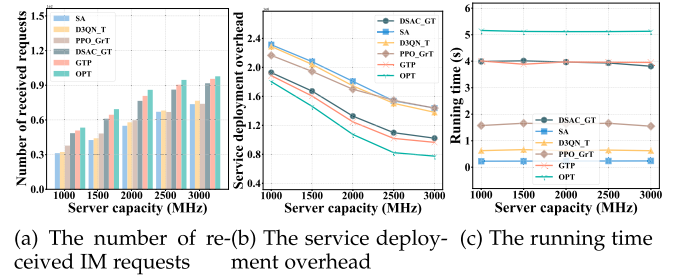


Fig. 5. The performance of different deployers by changing server capacity from 1,000 to 3,000.

value from 1,000 to 3,000. Other parameters are configured as specified in **Set 1** of Table III. As shown in Fig. 5(a), the number of requests received for all algorithms gradually rises with increasing server capacity. Additionally, the expanded capacity allows deployment locations to be selected from more candidate servers with lower costs or closer locations, leading to a downward trend in deployment overhead. As illustrated in Fig. 5, the GTP algorithm incurs a slightly higher overhead (4%~20%) than OPT but remains 3%~6% lower than DSAC while accepting 4%~6% more requests. In contrast, SA and PPO\_GrT exhibit the highest overhead, exceeding the optimal by at least 50%. In addition, even in small-scale scenarios, OPT, as an exhaustive search method, has significantly higher running time due to its high computational complexity, as shown in Fig. 5(c). In contrast, GTP, a learning-based algorithm, offers much faster inference for real-world applications.

**Server cost:** Fig. 6 demonstrates the influence of the server computation cost  $c_j$  on the proposed algorithm by adjusting its average value from 1.5 to 9.5. Other parameters are configured as specified in **Set 2** of Table III. As shown in Fig. 6(a), the number of accepted requests for all algorithms remains stable as costs increase, indicating that cost variations do not impact the request-handling capacity of algorithms. However, since deployment overhead is tied to distance and computing expenses, the overhead of all algorithms increases with rising costs. Among them, the GTP algorithm incurs slightly higher overhead than OPT, but at least 7%~14% lower than other comparison algorithms and accepts 6%~9% more requests.

**Overhead coefficient:** Fig. 7 presents the effect of the weighted overhead coefficient  $\epsilon_{oc}$  on a series of GTP algorithm by varying it from 0.3 to 0.7. Other parameters are configured

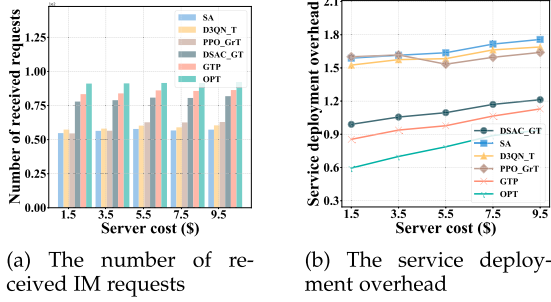


Fig. 6. The performance of different deployers by changing average server cost from 1.5 to 9.5.

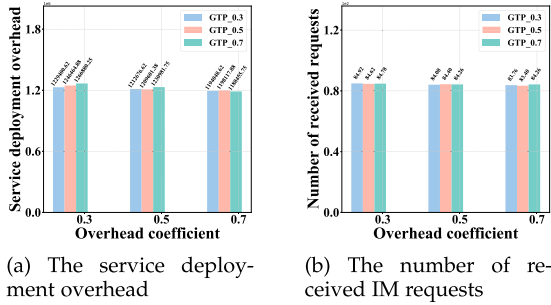


Fig. 7. The performance of different deployers by varying the overhead coefficient from 0.3 to 0.7.

as specified in **Set 3** of Table III. As shown in Fig. 7, the model trained with a specific attribute value often performs best under that attribute. For instance, when the coefficient is set to 0.3, GTP\_0.3 achieves the lowest overhead and the highest number of received requests. Conversely, when the coefficient is adjusted to 0.7, GTP\_0.7 outperforms the other GTP algorithms with different values of  $\epsilon_{oc}$ .

2) *Impact of Key Parameters on Intent Extractor:* We then trained the intent extractor over 20 training steps, using ‘delay’ and ‘cost’ as keywords for server providers. The detailed evaluation results are presented as follows.

These keywords were expanded into 200 intent prompts, respectively. Fig. 8(a) demonstrates that as the number of training steps increases, the ability of intent extractor to recognize provider intent improves progressively, achieving 100% accuracy over time. Although the algorithm is designed for binary classification using two keywords, it can be extended to handle multi-classification, as shown in Fig. 8(b), and open-world classification problems by incorporating additional keywords. Such extensions, however, are beyond the scope of this study.

3) *Impact of Key Parameters on Routers:* In the following, we trained all routers with a random seed of 123 and a training step of  $1 \times 10^3$ , and the detailed evaluation results are given as follows.

**Train process:** For different routers, Fig. 9 depicts the trend on total rewards, number of accepted requests, and optimization goal during model training. As shown in Fig. 9(a), as the number of training steps increases, the rewards for all routers rise initially and then level off. Among them, OPT consistently

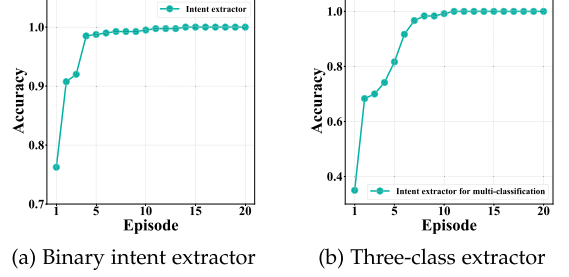


Fig. 8. The accuracy of intent extractors.

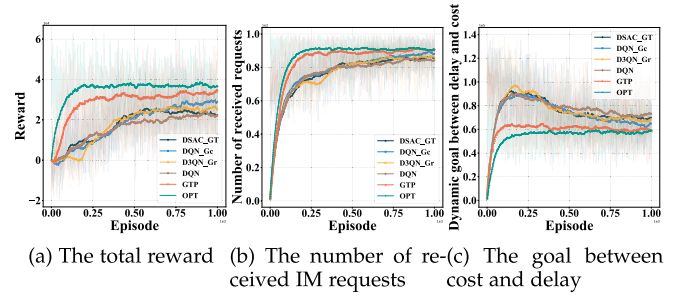


Fig. 9. The performance of routers during model training.

delivers the best performance due to its prior knowledge of the optimization goal, followed by GTP. In comparison, the DSAC\_GT, DQN\_Gc, DQN\_Gc, and DQN\_Gc algorithms result in lower rewards than these two, where DQN exhibits the poorest performance due to its lack of feature extraction capability, while models incorporating graph convolutional networks deliver better results. Additionally, the performance ranking of the above algorithm, in terms of the number of requests received and the optimization goals of service cost and delay, aligns with the performance trend indicated by the reward, as shown in Fig. 9(b) and (c).

Then, we analyzed the impact of key parameters, including provider intent, link delay, link cost, bandwidth capacity, and the data scaling coefficient, on the performance of the algorithms.

**Provider intent:** Fig. 10 reveals the impact of the keyword of provider intent prompt  $ip$  on the proposed routers by varying it from “delay” to “cost”. Other parameters are configured as specified in **Set 4** of Table III. As shown in Fig. 10, around the third time window, the shift in provider intent from minimizing delay to cost led to significant changes in the length of the multicast tree (namely, the number of links in a tree) and the optimization goal, while the number of requests received remained stable. This shows that algorithms can accurately recognize the provider intent and adapt to changes without impacting key performance metrics, and the final performance value depends on the attribute of the optimal action selected based on the current intent. Specifically, GTP performs slightly worse than the optimal OPT in terms of dynamic goals (2%~7%) and received requests (about 1.1%), but it accepts 1% more requests at 5%~10% lower goal than DSAC\_GT. However, the OPT

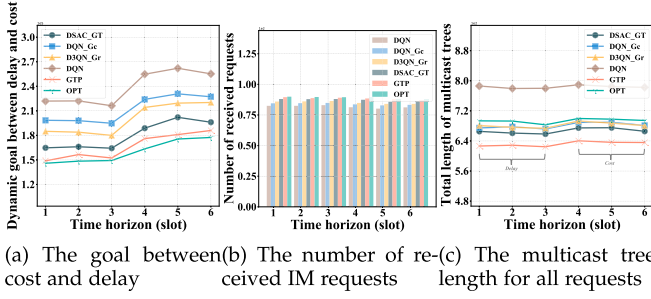


Fig. 10. The performance of different routers by changing provider intent from “delay” to “cost”.

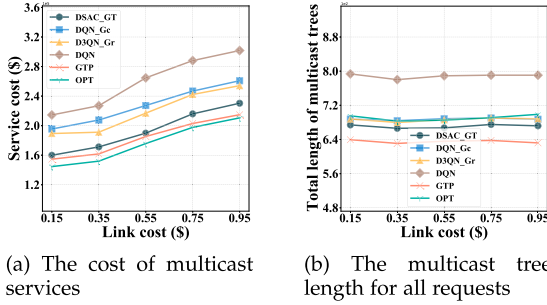


Fig. 11. The performance of different routers by varying the average link cost from 0.15 to 0.95.

algorithm first selects the shortest path between the primary and backup services, as well as source user, then connects the service to other target users, resulting in a longer multicast tree (about 9%) compared to GTP.

**Link cost:** Fig. 11 illustrates the impact of link cost  $c_l$  on the proposed algorithms by varying its average value from 15 to 95, with the provider intent set to “cost”. Other parameters are configured as specified in **Set 5** of Table III. As shown in Fig. 11(a), the total service cost of all algorithms rises with increasing link costs. While GTP incurs a slightly higher cost than OPT (up to 6%), it remains substantially lower than other baseline algorithms, achieving a cost reduction of up to 20.9% compared to DQN. Fig. 11(b) demonstrates that link cost has minimal impact on multicast tree length. Notably, the multicast tree length of GTP is at least 4.7% shorter than that of other algorithms. Similarly to computation cost, link cost also has little effect on the number of requests received. Thus, it is recommended to configure  $c_l$  within a mid-to-high range (0.55–0.95) to balance cost efficiency and performance. Avoid low values (0.15–0.55), where GTP’s cost advantage is less pronounced.

**Link delay:** Fig. 12 shows the effect of link delay  $D_l$  on the proposed algorithms by adjusting its average value from 15 to 95, while setting the provider intent to prioritize ‘delay’. Other parameters are configured as specified in **Set 6** of Table III. As illustrated in Fig. 12, GTP accepts 0.6% fewer requests than OPT and incurs up to 3% higher service (multicast) latency; however, it accepts 1%~1.7% more requests than DSAC\_GT while reducing overall service (multicast) latency by 7%~12%.

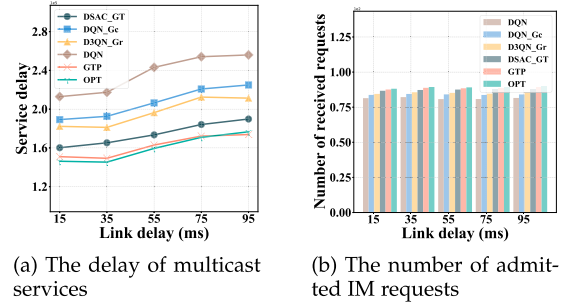


Fig. 12. The performance of different routers by varying the average link delay from 15 to 95.

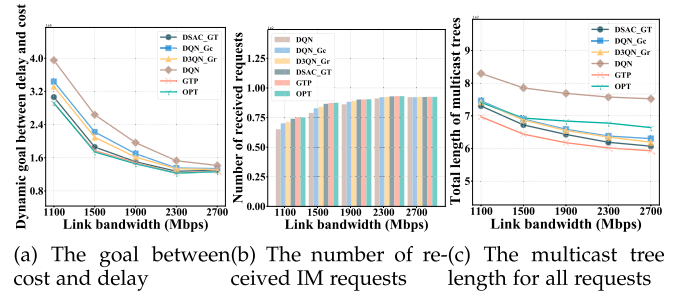


Fig. 13. The performance of different routers by varying the average bandwidth from 1,100 to 2,700.

Note that the influence of link delay on algorithm performance exhibits a similar pattern to that of link cost and computation cost. In particular, the service delay for all algorithms rises as the link delay increases, while the multicast tree length and the number of requests received remain consistent. Similarly, it is recommended to configure  $D_l$  within a medium to high range (55–95), as GTP performs better in relatively congested network environments.

**Bandwidth capacity:** Fig. 13 presents the impact of bandwidth capacity  $B_l$  on the proposed routers by varying its average value from 1,100 to 2,700, while the provider intent is randomly extracted. Other parameters are configured as specified in **Set 7** of Table III. As shown in Fig. 13, as the link bandwidth increases, all algorithms are able to accommodate more users while utilizing shorter multicast tree lengths. Specifically, while GTP performed marginally worse than OPT, achieving up to 1.7% more total delay and cost, it outperformed other algorithms, achieving at least 1.4% less than the closest competitor. This is because the GTP algorithm demonstrates superior performance compared to the other algorithms in terms of multicast tree length (at least 6%), and shorter trees can accommodate more requests for a network as they utilize fewer resources. In addition, the performance eventually stabilizes as the limiting constraint shifts from bandwidth to the total number of arrival requests within each time window. In conclusion, GTP is relatively insensitive to link bandwidth variations, maintaining stable performance across a wide range of bandwidth configurations. Therefore, bandwidth can be flexibly allocated based on practical requirements.



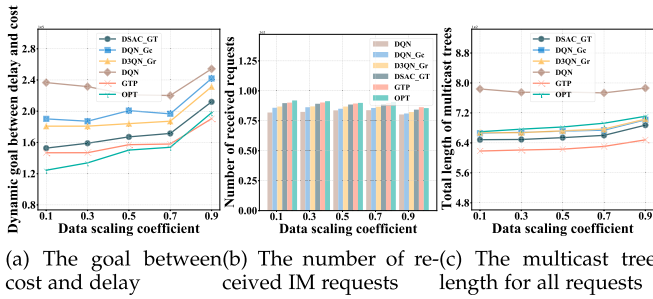


Fig. 14. The performance of different routers by varying the data scaling coefficient from 0.1 to 0.9.

**Data scaling coefficient:** Fig. 14 presents the impact of data scaling coefficient  $\Delta_k$  on the proposed algorithms by varying its value from 0.1 to 0.9, while the service intent is randomly extracted. Other parameters are configured as specified in **Set 8** of Table III. With the growth of the coefficient value, the total bandwidth consumption across the network increases. This leads to a slight reduction in the number of requests received by all algorithms, while both the optimization objective and the multicast tree length increase, as shown in Fig. 14. Additionally, in comparison with Fig. 13, it can be seen that the variations in the coefficient have less impact on the key performance metrics of the algorithms than fluctuations in bandwidth. ALL in all, GTP can tolerate moderate data fluctuations. For efficient information sharing, the  $\Delta_k$  should be set for scenarios where the data volume change before and after service processing is not too drastic, ideally more than 0.5.

## VI. CONCLUSION

In this paper, we first formulated a novel intent-driven reliable information sharing problem in multi-tenant metaverse networks, which explicitly considers network dynamics and service reliability. We then proposed a learning-based system framework, named GTP, combining proximal policy optimization reinforcement learning with feature extraction networks, including graph attention network and gated recurrent unit, and a Transformer encoder for multi-feature comparison, which integrates service deployments, natural-language intent extractions, and multicast routing for online information sharing. Finally, we conducted theoretical analysis and simulations to evaluate the performance of the proposed algorithms. Simulation results demonstrate that the proposed algorithms efficiently reduce the deployment overhead, service costs, and service delays, while improving system throughput. Looking ahead, intent-driven information sharing and network management in the metaverse is still in its early stages. The rapid advancement of large language models presents new opportunities to develop more autonomous and proactive agents for end-to-end management. Furthermore, integrating such agents with reinforcement learning to enhance tool use and reasoning capabilities may enable closed-loop optimization across user, service, and resource layers, advancing toward fully intelligent multi-tenant metaverse networks.

## REFERENCES

- [1] F. Chai, Q. Zhang, H. Yao, X. Xin, R. Gao, and M. Guizani, "Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT," *IEEE Trans. Veh. Technol.*, vol. 72, no. 6, pp. 7783–7795, Jun. 2023.
- [2] A. Collet, A. Banchs, and M. Fiore, "LossLeap: Learning to predict for intent-based networking," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2022, pp. 2138–2147.
- [3] A. Collet, A. Bazco-Nogueras, A. Banchs, and M. Fiore, "AutoManager: A meta-learning model for network management from intertwined forecasts," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2023, pp. 1–10.
- [4] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (GRU) neural networks," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 1597–1600.
- [5] H. Du et al., "Diffusion-based reinforcement learning for edge-enabled AI-generated content services," *IEEE Trans. Mobile Comput.*, vol. 23, no. 9, pp. 8902–8918, Sep. 2024.
- [6] H. Du, J. Wang, D. Niyato, J. Kang, Z. Xiong, and D. I. Kim, "AI-generated incentive mechanism and full-duplex semantic communications for information sharing," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 9, pp. 2981–2997, Sep. 2023.
- [7] K. Han et al., "A survey on vision transformer," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 1, pp. 87–110, Jan. 2023.
- [8] H. Hosseini Nasab and F. Mobasheri, "A simulated annealing heuristic for the facility location problem," *Int. J. Math. Model. Numer. Optim.*, vol. 4, no. 3, pp. 210–224, 2013.
- [9] M. Imran, M. N. Ali, M. S. U. Din, M. A. U. Rehman, and B.-S. Kim, "An efficient communication and computation resources sharing in information-centric 6G networks," *IEEE Internet Things J.*, vol. 11, no. 16, pp. 27275–27294, Aug. 2024.
- [10] A. S. Jacobs et al., "Hey, Lumi! using natural language for intent-based network management," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, USENIX Association, Jul. 2021, pp. 625–639. [Online]. Available: <https://www.usenix.org/conference/atc21/presentation/jacobs>
- [11] A. Leivadreas and M. Falkner, "A survey on intent-based networking," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 625–655, 1st Quart. 2023.
- [12] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, "Serverless computing: State-of-the-art, challenges and opportunities," *IEEE Trans. Services Comput.*, vol. 16, no. 2, pp. 1522–1539, Mar./Apr. 2023.
- [13] G. Liu, M. Bilal, Q. Wang, and X. Xu, "Congestion-aware path planning with vehicle-road cooperation in AtoV," *IEEE Internet Things J.*, vol. 11, no. 24, pp. 39–626–39 636, Dec. 2024.
- [14] H. Ma, Z. Zhou, X. Zhang, and X. Chen, "Toward carbon-neutral edge computing: Greening edge AI by harnessing spot and future carbon markets," *IEEE Internet Things J.*, vol. 10, no. 18, pp. 16637–16649, Sep. 2023.
- [15] Y. Ma, W. Liang, J. Wu, and Z. Xu, "Throughput maximization of NFV-enabled multicasting in mobile edge cloud networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 393–407, Feb. 2020.
- [16] E. R. Magsino and I. W.-H. Ho, "An enhanced information sharing roadside unit allocation scheme for vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 9, pp. 15462–15475, Sep. 2022.
- [17] Y. Qiu et al., "Spotlighter: Backup age-guaranteed immersive virtual vehicle service provisioning in edge-enabled vehicular metaverse," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 13375–13391, Dec. 2024.
- [18] Y. Qiu, M. Chen, W. Liang, L. Ai, D. Niyato, and G. Wei, "Privacy-enhanced healthcare monitoring service refreshment in human digital twin-assisted fabric metaverse," *IEEE Trans. Mobile Comput.*, early access, Jun. 23, 2025, doi: 10.1109/TMC.2025.3582084.
- [19] Y. Qiu et al., "Reliable or green? Continual individualized inference provisioning in fabric metaverse via multi-exit acceleration," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11449–11465, Dec. 2024.
- [20] Y. Qiu, J. Liang, V. C. M. Leung, X. Wu, and X. Deng, "Online reliability-enhanced virtual network services provisioning in fault-prone mobile edge cloud," *IEEE Trans. Wireless Commun.*, vol. 21, no. 9, pp. 7299–7313, Sep. 2022.
- [21] H. Ren et al., "Efficient algorithms for delay-aware NFV-enabled multicasting in mobile edge clouds with resource sharing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 9, pp. 2050–2066, Sep. 2020.
- [22] H. Sami et al., "The metaverse: Survey, trends, novel pipeline ecosystem & future directions," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 4, pp. 2914–2960, 4th Quart. 2024.

- [23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
- [24] X. Shang, Y. Huang, Y. Mao, Z. Liu, and Y. Yang, "Enabling QoE support for interactive applications over mobile edge with high user mobility," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2022, pp. 1289–1298.
- [25] X. Shang, Y. Mao, Y. Liu, Y. Huang, Z. Liu, and Y. Yang, "Online container scheduling for data-intensive applications in serverless edge computing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2023, pp. 1–10.
- [26] F. Tian, X. Zhang, J. Liang, and Z. Yang, "Bidirectional service function chain embedding for interactive applications in mobile edge networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 3964–3980, May 2024.
- [27] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.
- [28] D. Wu et al., "NetLLM: Adapting large language models for networking," in *Proc. ACM SIGCOMM Conf. (ACM SIGCOMM)*, New York, NY, USA: ACM, 2024, pp. 661–678.
- [29] M. Yang et al., "Deep reinforcement learning-based joint caching and routing in AI-driven networks," *IEEE Trans. Mobile Comput.*, vol. 24, no. 3, pp. 1322–1337, Mar. 2025.
- [30] M. Ye, C. Zhao, P. Wen, Y. Wang, X. Wang, and H. Qiu, "DHRL-FNMR: An intelligent multicast routing approach based on deep hierarchical reinforcement learning in SDN," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 5, pp. 5733–5755, Oct. 2024.
- [31] M. Zawish et al., "AI and 6G into the metaverse: Fundamentals, challenges and future research trends," *IEEE Open J. Commun. Soc.*, vol. 5, pp. 730–778, 2024.
- [32] Y. Zhang, W. Liang, Z. Xu, X. Jia, and Y. Yang, "Profit maximization of delay-sensitive, differential accuracy inference services in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 24, no. 7, pp. 6209–6224, Jul. 2025.
- [33] K. Zhao, Z. Zhao, Z. Wang, and H. Zhang, "Routing algorithm design based on deep reinforcement learning and GNN," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2024, pp. 1–6.
- [34] M. Zhu, F. He, and E. Oki, "Multiple backup resource allocation with workload-dependent failure probability," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2020, pp. 1–6.