OMNIS: Semantic RAN Slicing via Dynamic Split Neural Networks

Langtian Qin†, Ian Andrew Harshbarger†, Leïla Nasraoui⋆, Carla Fabiana Chiasserini‡, Marco Levorato†
†Donald Bren School of Information and Computer Science, University of California Irvine, USA
⋆National School of Computer Science (ENSI), University of Manouba, Tunisia
‡Politecnico di Torino and CNIT, Italy
Email : {langtiq, iharshba}@uci.edu, leila.nasraoui@supcom.tn, carla.chiasserini@polito.it, levorato@uci.edu

***Abstract*—Edge computing enables resource-constrained devices to execute machine learning applications via task offloading. To this aim, radio access network (RAN) slicing is instrumental to provide the necessary network resources. However, current RAN slicing approaches rely on static computing models, thereby constraining their ability to leverage the dynamic semantic data representation capabilities enabled by recent neural architectures. In this paper, we propose OMNIS, a semantic RAN slicing framework for edge computing built on dynamic split neural models. OMNIS embeds a dynamic form of neural compression paired with adaptive data encoding for task offloading, enabling flexible communication payload and computing options. We explicitly study the interplay between neural compression and information quantization in computer vision tasks and design a novel "Box" quantization scheme that improves resiliency to bit errors as a function of compression rate. Considering partial observability and differing objectives of mobile devices and the edge server, we formulate neural gate control and resource slicing optimization problems and solve them via multi-agent contextual multi-armed bandits and convex optimization algorithms. Experimental results show that OMNIS improves inference accuracy by up to 22.85% and reduces quality of service violations by up to 10×. The evaluation code is available at https://github.com/qlt315/OMNIS.**

***Index Terms*—Edge computing, machine learning, resource allocation, wireless communications**

I. Introduction

The rapid rise of machine learning (ML)-powered applications, including real-time video analytics, autonomous driving, and augmented reality, has introduced latency-sensitive and computationally intensive workloads, often involving complex neural networks and large data volumes [1]. As a result, resource-constrained mobile devices (MDs) face challenges in meeting quality of service (QoS) requirements. By offloading computing-intensive ML tasks to edge servers (ES), edge computing has emerged as a paradigm to support real-time inference by significantly reducing task execution latency [2]. In this context, radio access network (RAN) slicing plays a critical role in supporting edge computing by enabling the flexible and efficient provisioning of communication and computing resources. However, most RAN slicing frameworks for edge computing directly transmit the raw sensor data and perform inference on the ES [3], [4]. Such an approach may require large data rates to make edge computing functional.

This work is funded in part by SNS JU under the EU's HE programme under the MultiX project Grant Agreement No. 101192521. Part of this work is also supported by the US NSF under grant CCF 2140154.

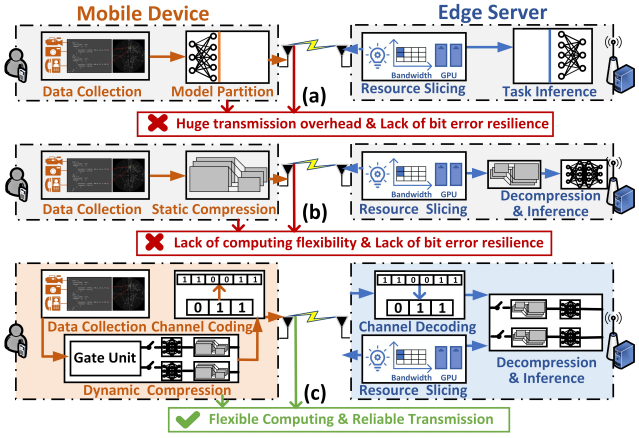

Fig. 1: Comparison of existing RAN slicing frameworks for edge computing and OMNIS's approach: (a) Offloading with neural network partitioning [2]; (b) Offloading with static data compression [7], [8]; (c) OMNIS.

For example, in image-based tasks using the COCO dataset [5], transmitting JPEG-compressed images can require hundreds of Kilobits per frame. Some recent frameworks [2] – see Fig. 1(a) – *split* deep neural network architectures and transmit the tensors produced by a set of neural layers executed by the MD. However, this approach may result in excessive bandwidth consumption as well; for instance, in ResNet-50 [6] the intermediate feature map at the "conv3" layer has a size of 3.1 MB when stored in FP32 format. The works in [7], [8] exemplify two other classes of contributions based on semantic communications – see Fig. 1(b). Specifically, [7] relies on static neural networks, such as joint source-channel coding via autoencoders, which lacks adaptability under fluctuating bit error rates (BER), as different BER levels can degrade inference performance to varying degrees. [8], instead, introduces an adjustable compression factor, allowing for dynamic semantic control over data reduction. However, none of these methods explicitly define how to implement such flexible compression, leaving a gap in practical deployment.

To address these issues, we propose a novel semantic RAN slicing framework called OMNIfacet Slicing (OMNIS). OMNIS connects semantic RAN slicing to a new generation of neural models: dynamic neural networks, an umbrella that includes multi-branch architectures. Notably, in this paper

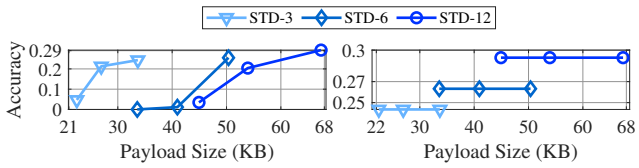

Fig. 2: Accuracy vs. payload size for different quantization methods at SNR equal to 3 dB (left) and 10 dB (right).

we develop a new class of dynamic neural networks that focuses flexibility in how the input data is neurally compressed into a compact latent space and quantized[1]. At a high level, on the MD side, the neural model we build has branches that correspond to supervised in-model neural compression encoder-decoder structures chained with a set of quantization options – see Fig. 1(c). The different branches, whose output is encoded with low-density parity check (LDPC) codes to obtain a specific data payload, offer a tradeoff between compression on one hand and resilience to bit errors and task performance on the other. After transmission over the wireless channel, the data is then decoded, decompressed at the ES, and then used for inference. *The result is an extremely flexible semantic data representation and processing pipeline that can boost the overall system performance and efficiency when combined with RAN resource slicing*.

To better highlight the impact of data compression on inference performance under different transmission conditions, we conducted an illustrative experiment. Fig. 2 shows the semantic segmentation accuracy, trained on the COCO dataset [5], versus transmission payload size under varying values of signal to noise ratio (SNR). The neural network model used to generate the results in Fig. 2 is detailed in Sec. IV, while the parameter settings are given in Sec. VI. Notably, we use a split deep neural network [9] where the model is modified to include an encoder/decoder structure neurally compressing the data into a latent space. The tensor is then quantized using the "standard" (STD) quantization method, meaning that we quantize the 32-bit values tensor representation using a direct normalization and conversion to 8-bit integer values [10]. 3, 6, and 12 denote the number of feature channels output from the neural encoder, with higher numbers indicating lower quantization levels. We apply LDPC coding with coding rates (payload size/total data size) $\{0, 2/3, 5/6\}$ to the compressed output of the neural encoder embedded in the models. Thus, the payload size is determined by the number of feature channel, the quantization levels, and the LDPC coding rate.

This preliminary evaluation shows that accuracy increases with data payload up to a point, after which it plateaus. This is more evident at high SNR, where adding more bits increases the transmitted data, but accuracy remains constant. Both MD's channel quality and payload size significantly impact accuracy. Poor channel conditions increase BER, leading to erroneous data for inference at the ES, while larger payload sizes improve accuracy but clearly increase transmission overhead. These results thus show that flexibility in how data is compressed is essential to obtain functional – both communication and computing –pipelines.

Our contributions can thus be summarized as follows:

- We propose OMNIS, a semantic RAN slicing framework for edge computing. OMNIS exploits a dynamic split DNN architecture that can produce flexible compressed data representations designed to support inference tasks. The neural network branches include advanced supervised neural encoder-decoder structures, paired with strategies for quantization of the data latent representation.
- In this context, we propose a new quantization approach to improve the tradeoff between compression on one hand and resilience to bit errors and task performance on the other, in channel regimes where current strategies are ineffective. To the best of our knowledge, this is the first paper to consider such a tradeoff in the design of dynamic split DNNs and semantic communications.
- To balance transmission efficiency and inference accuracy, we formulate two optimization problems for MDs and ES (resp.), considering their partial view of the system and different objectives. MDs aim to maximize their own task accuracy by controlling the dynamic split DNN, while the ES seeks to maximize the minimum accuracy across MDs for fairness. To solve these intractable problems, we design a multi-agent distributed optimization framework, where each MD acts as a contextual multi-armed bandit agent using Bayesian optimization to select the best neural branch, and the ES allocates resources via block coordinated descent (BCD).
- We perform an extensive evaluation demonstrating the effectiveness of OMNIS. Compared to state-of-the-art RAN slicing schemes, OMNIS improves inference accuracy by up to 22.85% while reducing the QoS constraint violation probability by up to 10x.

II. Related Work

Mobile Edge Computing. The most recent works aim at optimizing task offloading to the edge, via binary or partial offloading. In binary offloading, tasks are indivisible, requiring full local execution or complete offloading [11]. Partial offloading (or split computing) splits inference, executing a portion of it locally and offloading the rest. [12]. Some studies jointly optimize task offloading and resource slicing [13], but adopt a binary offloading model, thus limiting the flexibility in determining the communication and computing model. Conversely, partial offloading approaches often assume simplistic and non-adaptive task partition strategies. OMNIS presents an explicit connection between dynamic split neural networks, compression, encoding and resource slicing, ensuring robust and deployable offloading across diverse network conditions and application demands.

RAN Slicing. It virtualizes network resources into isolated partitions for flexible allocation. Examples of proposed strategies include [14], which leverages deep reinforcement learning (DRL) for content caching and mode selection in hotspot and vehicular scenarios. [15] models slicing as a congestion

[1]We use *"neural compression"* for the neural encoding process and *"compression"* to denote the combined process of neural encoding and quantization.

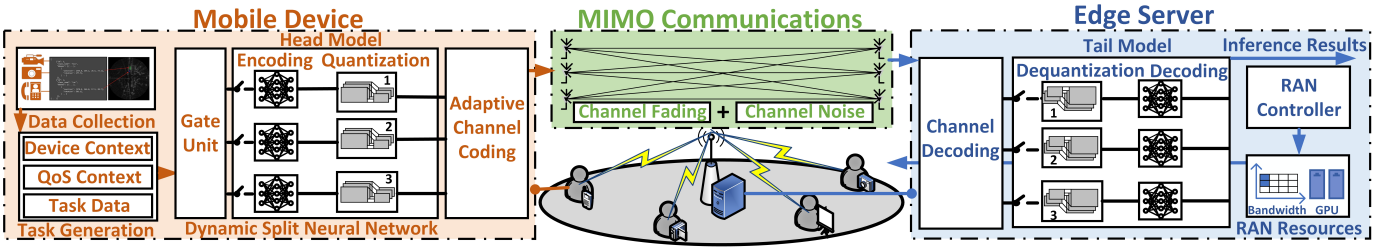

Fig. 3: System model of OMNIS.

game with a unique Nash equilibrium (NE), and designs two fully distributed algorithms to converge to the NE without exposing sensitive tenant information. Also, [16] first identifies MDs with satisfiable QoS and then jointly optimizes slice association and bandwidth allocation to minimize resource consumption. However, existing frameworks are designed for static neural networks, thus failing to leverage important synergies between the adaptation capabilities of ML tasks and resource allocation. OMNIS overcomes these limitations by integrating dynamic neural networks and jointly optimizing RAN resource allocation and DNN branch control, enabling personalized provisioning and task-specific customization.

Semantic Communications. Semantic communications leverage ML-driven data compression to extract and transmit only the most relevant information. For instance, VQ-DeepSC [17] uses ML-based transceivers and adversarial training to extract multi-scale semantic features for image transmission. Similarly, [18] introduces RL-based adaptive semantic coding to preserve task-relevant information while discarding less critical data. [19], instead, formalizes a rate-distortion tradeoff, proposing dynamic neural networks for variable-length feature encoding depending on channel conditions. SEM-O-RAN [8], the closest work to ours, enables semantic RAN slicing for task offloading. Through a greedy algorithm, it optimizes data compression and the use of networking and computing resources. However, SEM-O-RAN does not specify the exact approach for data compression, nor does it account for the impact of wireless transmission errors on accuracy. Other works use static neural networks, whereas OMNIS employs dynamic neural networks that flexibly balance compression ratio and accuracy, enhancing resource utilization and efficiency, especially on unreliable radio channels.

III. OMNIS System Model

The system model of OMNIS, depicted in Fig. 3, includes an edge computing-enabled RAN, consisting of a multi-antenna base station integrated with an ES and a set $\mathcal{M}=\{1,\dots,M\}$ of multi-antenna MDs distributed across the coverage area. Each MD $m\in\mathcal{M}$ and the ES are equipped with ρ^m and ρ^e antennas (resp.). The system operates in time slots indexed by $t=0,\dots T$. In each time frame, composed of multiple time slots, each MD processes a computer vision job containing multiple inference tasks, where each task corresponds to processing a segment of data and performing inference within a single time slot. For instance, in the video surveillance application, the inference job of each MD is to process a video stream, with each task corresponding to analyzing a single video frame. The MD generates the ML inference task based on data collected from cameras. The task encapsulates various contextual information, including the MD's QoS requirements and preferences, and its computational capacity, channel conditions, and a summary of the data size to be processed.

In OMNIS, the MDs use a dynamic split DNN composed of multiple branches, which are dynamically selected by a gate unit. Each branch includes a dynamic split DNN sub-model designed with an advanced encoder-decoder architecture. As detailed later, the characteristics of the encoder-decoder control the tradeoff between compression gain on the one hand and resilience to bit errors and task performance on the other. Each branch–or sub-model –consists of two parts: the head model on the MD side and the tail model on the ES side (detailed in Sec. IV). The head model is responsible for extracting semantic features from the raw data, compressing the features and then performing channel coding before transmission to the tail model on the ES via the multiple-input-multiple-output (MIMO) wireless channel. The tail model realizes the inverse process of the head model and generates the inference results, which are then returned to the MDs.

At the MD side, an optimization engine controls the gate unit of the dynamic split DNN model, using the task context as input. This enables the use of sub-models with varying compression (i.e., neural compression and quantization) capabilities. At the ES, a RAN controller (i) dynamically allocates bandwidth to MDs for their sending head model outputs, and (ii) assigns computing resources to support the execution of their tail models, optimizing both communication and computation efficiency. The controller is also responsible for adjusting the MDs' channel coding rate. Thus, as detailed in Sec. V, *OMNIS enables gate unit control and RAN resource slicing in a distributed and context-aware manner*. The main OMNIS components are described below.

A. Mobile Device Computing Model

In each time slot t, the inference task generated by MD $m\in\mathcal{M}$ has specific QoS requirements, denoted as $\{\tau_m^{th}(t), E_m^{th}(t), \omega_m^{\mathrm{t}}(t), \omega_m^{e}(t)\}$, where $\tau_m^{th}(t)$ and $E_m^{th}(t)$ are the expected latency and energy consumption targets, respectively. The preference weights $\omega_m^{\mathrm{t}}(t)$ and $\omega_m^{e}(t)$ reflect MD m's latency and energy priorities, e.g., a drone may prioritize low latency with a higher $\omega_m^{\mathrm{t}}(t)$, while an IoT sensor may prioritize energy efficiency with a higher $\omega_m^{e}(t)$. Due to task variability, strict adherence to these targets may not always be

possible, so violations are allowed to ensure task completion. The gate unit, based on the optimization engine, selects a specific branch (detailed in Sec. V), then feeds the raw data X into the head model of the branch, where it is neurally compressed and quantized. The quantized bit stream $\boldsymbol{x}_m^c(t)$ then undergoes channel coding and modulation. Based on the gate unit, OMNIS enables dynamically adapting the quantization method $\beta_m(t)$ and the feature channel size $\lambda_m(t)$, based on each MD's wireless channel conditions and task requirements. The local computing latency of MD m in time slot t is given by: $\tau_m^l(t)=\psi_m^h(t)/(f_m^l c_m^l \chi_m^l)$, where $\psi_m^h(t)$ is the FLOPs of the head model, f_m^l, c_m^l, and χ_m^l represent the GPU frequency, number of GPU cores, and FLOPs per cycle per core of MD m, respectively. Given the GPU-dependent power consumption coefficient, ζ_m^l, the local energy consumption is: $E_m^l(t)=\zeta_m^l (f_m^l)^3 \cdot \tau_m^l(t)=\zeta_m^l \left(f_m^l\right)^2 \psi_m^h(t)/c_m^l \chi_m^l$.

B. Transmission Model

To ensure reliable radio transmission, the compressed and quantized data stream $\boldsymbol{x}_m^c(t)$ undergoes channel encoding (e.g., using LDPC coding), which adds redundancy to reduce BER by detecting and correcting errors caused by the wireless channel. After channel encoding, modulation maps the bits onto symbols for transmission over the MIMO channel. In OMNIS, the channel coding rate $\phi_m(t)\in\boldsymbol{\Phi}$ is dynamically adjusted based on channel conditions, task requirements, and quantization parameters of MD $m\in\mathcal{M}$. This flexibility allows OMNIS to balance transmission reliability and efficiency.

Let $\boldsymbol{x}_m^{\mathrm{t}}(t)$ be the transmitted modulated bit stream of MD m and $\boldsymbol{H}_m(t)$ the wireless channel matrix between the ES and the MD m in time slot t. We consider a flat fading channel throughout each slot. Imperfect channel state information (CSI) is assumed due to noise, estimation errors, and channel variations. The estimated channel matrix is $\hat{\boldsymbol{H}}_m(t)=\boldsymbol{H}_m(t)+\boldsymbol{H}_m^e(t)$, where $\boldsymbol{H}_m^e(t)$ is the error matrix, modeled as zero-mean Gaussian noise with variance σ^h. To mitigate intra-cell interference, the ES divides the bandwidth into orthogonal subchannels using orthogonal frequency-division multiple access (OFDMA). Let B be the total bandwidth and $b_m(t)$ that allocated to MD m in time slot t, then MD m's achievable rate is: $\delta_m(t)=\log\det\left(\boldsymbol{I}_{\rho^e}+\frac{\boldsymbol{H}_m(t)\boldsymbol{H}_m^{\mathsf{H}}(t)}{(\sigma^n)^2}\right)$. The transmission latency $\tau_m^{\mathrm{t}}(t)$ is $\tau_m^{\mathrm{t}}(t)=\frac{d_m(t)}{b_m(t)\delta_m(t)}$, where $d_m(t)$ is the data size of $\boldsymbol{x}_m^{\mathrm{t}}(t)$. Given MD m's transmission power, $p_m(t)$, its transmission energy is given by $E_m^{\mathrm{t}}(t)=p_m(t)\tau_m^{\mathrm{t}}(t)$.

C. Edge Server Computing Model

Upon reception at the ES, the signal undergoes demodulation and channel decoding, yielding the estimated bit stream $\hat{x}_m^c(t)$. This stream is then processed through the dequantization module of the selected tail model, composed of the neural decoder, to reconstruct the features, and the analysis module to produce the final inference result. Let F^e represent the total computing capacity of the ES in each time slot, expressed as GPU frequency, and let $f_m^e(t)$ denote the GPU frequency allocated to MD m during time slot t. The edge computing latency of MD m in time slot t can then be written as: $\tau_m^e(t)=\frac{\psi_m^{\mathcal{T}}(t)}{f_m^e(t)c^e\chi^e}$, where $\psi_m^{\mathcal{T}}(t)$ is the FLOPs of the tail model corresponding to the head model selected by MD m, c^e and χ^e are the number of GPU cores and number of FLOPs per cycle per core of the ES, respectively. Similarly, the energy consumption is: $E_m^e(t)=\frac{\zeta^e(f_m^e(t))^2\psi_m^{\mathrm{t}}(t)}{c^e\chi^e}$, where ζ^e is the power consumption coefficient of the ES. We observe a trade-off in the allocation of computing resources: increasing the GPU frequency reduces processing latency, but this comes at the cost of higher energy consumption. The final inference result is then transmitted to MD m; we denote by $\xi_m(t)$ the obtained inference accuracy. Notice that, since the data size of the tail model output is significantly smaller than the input, the downlink transmission latency is negligible in the computation of the total latency.

IV. Dynamic Split DNN Model

The architecture of the proposed dynamic split DNN is shown in Fig. 3. It is a gated DNN in which a gate controls data routing and activates distinct model sections, referred to as "branches." As detailed later, gate decisions are produced by a component of the OMNIS distributed optimization engine. Each branch is a split-DNN submodel with L layers and a bottleneck inserted at a designated layer, partitioning the network into a head model $\mathcal{H}$ with l layers and a tail model $\mathcal{T}$ with $(L-l)$ layers. The head performs neural compression and quantization, converting tensor features into a compact bit-level representation, thereby trading task performance for reduced payload. Next, we describe the head model, the gate unit, and the tail model.

(i) Head Model. It resides at the MD and comprises the first l layers, acting as a neural encoder $\mathcal{H}(x)$ to extract semantic features from an input sample x. Following [10], the encoder serves as a neural compressor that maps x to an intermediate tensor z in a latent space. We do not impose an explicit loss directly on z; instead, z is learned end-to-end via backpropagation from the task/reconstruction objective applied in the tail model. A quantization module is then applied to z to generate a compact representation for transmission. As illustrated in Fig. 2 and confirmed by our results, quantization is essential to balance payload size with robustness to bit errors and overall task performance. **(ii) Tail Model.** It is deployed at the ES and performs the final inference. It mirrors the head by using two decoding stages: (i) an algorithmic dequantization module to recover the quantized tensor, and (ii) a neural decoder trained via teacher–student distillation [10] to map the received compressed features z to the target representation at the j-th layer of a larger, non-split teacher model, denoted by h_j. Specifically, the neural decoder $f_{\mathrm{dec}}(z)$ decompresses the input to reconstruct the intermediate feature h_j, after which the remaining layers of the tail are copied from the teacher model beyond layer j to complete inference. **(iii) Gate Unit.** The proposed architecture supports dynamic split-DNN configuration through contextual, optimization-driven gated selection. The gate determines which split sub-model (branch) is activated, thereby directly affecting key metrics

such as energy consumption, delay, and accuracy. In particular, it selects the quantization method $\beta_m(t)$ and the number of quantization channels $\lambda_m(t)$ from predefined sets to optimize overall system performance. Beyond existing options, we introduce *Box quantization*, which supports multiple quantization levels followed by source coding, enabling adaptive, per-context configuration before transmission. The proposed Box quantization scheme is described next.

The quantization method in Fig. 2, termed *standard* [10], converts 32-bit features to 8-bit values by min-max normalization followed by scaling of each element in the extracted semantic feature tensor x'. While simple, it yields a modest compression ratio, leading to a large payload and high transmission overhead. In contrast, entropy-based quantization [20] (e.g., Huffman/arithmetic coding) can compress the tensor into a much smaller bitstream, producing a compact representation z rather than x'. However, entropy coding is highly error-intolerant: a single bit error may corrupt symbol boundaries (frame shift), making it unsuitable for wireless edge links where transmission errors are common.

To balance compression and bit-error tolerance, we propose a resilient *Box quantization* scheme. We first apply standard quantization to x and then perform a learnable run-length encoding (RLE) to further reduce the payload while maintaining reliable dequantization. Specifically, the quantized tensor is flattened into a vector $\boldsymbol{x}^q$ and encoded into a byte stream of (value, frequency) pairs, where the first byte stores the value and the second byte stores the number of consecutive repeats. We set the bit depth to 8, limiting each run length to 255 to satisfy byte-size constraints. Since pairing values and frequencies (and the associated reshaping of $\boldsymbol{x}^q$) breaks gradient flow in the head model, we employ a neural decoder for reconstruction and optimize it with a reconstruction loss $\mathcal{L}^c(\boldsymbol{x}^q) = \upsilon\left(\sum_{i=1}^{|\boldsymbol{x}^q|-1}\left|x^q_{i+1}-x^q_i\right|\right)^2/(|\boldsymbol{x}^q|-1)$. The parameter υ is a dynamic compression scaling factor that favors compressibility for high values at the cost of performance, while its lower values prioritize reconstruction accuracy. The function $\mathcal{L}^c(\cdot)$ minimizes the differences between consecutive values in $\boldsymbol{x}$, promoting sequences of identical values that reduce encoding size. However, excessive compression can lead to $\boldsymbol{x}$ with uniform values, hindering accurate reconstruction of $\boldsymbol{x}$ at the tail model.

To illustrate the impact of quantization on inference performance, we evaluate accuracy versus payload size under different SNRs, as shown in Fig. 4. For Box quantization training, we set $\upsilon \in 0.15, 0.075, 0.0375$ for bottleneck channel sizes of 12, 6, and 3, respectively. We also include an entropy-based baseline ("ENT") that adopts a near-identical autoencoder architecture to the standard method but replaces the quantizer with advanced (partially differentiable) entropy coding, e.g., range asymmetric numeral systems (rANS) [20].

Fig. 4 provides two key observations. First, entropy quantization is extremely error-sensitive: even with high-rate LDPC protection, a single bit error can cause severe accuracy collapse, often approaching zero. Second, standard quantization

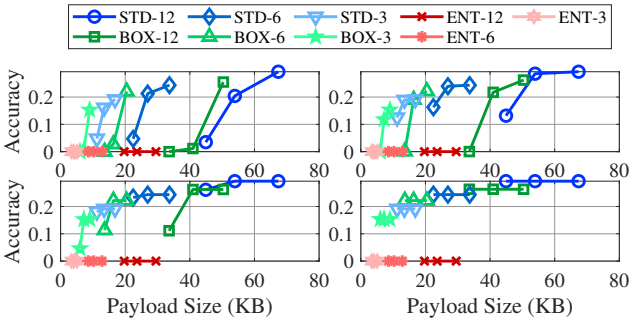

Fig. 4: Accuracy vs. payload size for different quantization methods at various SNRs: 3 dB (top-left), 5 dB (top-right), 7 dB (bottom-left), 10 dB (bottom-right).

is considerably more robust but requires the largest payload to achieve a given accuracy. Box quantization lies between these extremes, offering a more favorable accuracy–payload trade-off across SNRs. Since no single scheme dominates under all channel conditions and payload constraints, we next formulate an optimization problem to dynamically select the quantization strategy jointly with resource allocation.

V. OMNIS Optimization Framework

To achieve optimal system performance, gate unit control and RAN resource slicing should be jointly optimized based on task contexts and user channel conditions. Traditional centralized optimization algorithms adopted by other RAN slicing frameworks suffer from significant drawbacks, including a large solution space which leads to high computational complexity and a high risk of single points of failure. Also, the optimization objectives of MDs and the ES are inherently different. Typically, MDs act selfishly, viewing other MDs as competitors for limited resources and focusing solely on maximizing their own benefits. In contrast, the ES must consider overall network fairness, ensuring a balanced allocation of resources across all MDs. Due to these factors, formulating a global optimization problem is impractical. We thus formulate a gate unit control problem for each MD, and establish a RAN resource slicing problem for the ES. Let $\tau^s_m(t)=\tau^l_m(t)+\tau^{\mathrm{t}}_m(t)+\tau^e_m(t)$ be the total latency and $E^s_m(t)=E^l_m(t)+E^{\mathrm{t}}_m(t)+E^e_m(t)$ the total energy consumption. Define $\boldsymbol{\beta}=\{\beta_m(t), m\in\mathcal{M}, 0\le t\le T\}$, $\boldsymbol{\lambda}=\{\lambda_m(t), m\in\mathcal{M}, 0\le t\le T\}$, $\boldsymbol{\phi}=\{\phi_m(t), m\in\mathcal{M}, 0\le t\le T\}$, $\boldsymbol{b}=\{b_m(t), m\in\mathcal{M}, 0\le t\le T\}$, and $\boldsymbol{f}^e=\{f^e_m(t), m\in\mathcal{M}, 0\le t\le T\}$. In the gate unit control problem, each MD aims to maximize its long-term average inference accuracy under latency and energy constraints, which is formulated as:

$$\mathrm{P1}: \quad \max_{\boldsymbol{\beta},\boldsymbol{\lambda}} \frac{1}{T}\sum_{0\le t\le T}\xi_m(t) \tag{1a}$$

$$s.t \quad \tau^s_m(t) \le \tau^{\mathrm{th}}_m(t), 0\le t\le T, \tag{1b}$$

$$E^s_m(t) \le E^{\mathrm{th}}_m(t), 0\le t\le T, \tag{1c}$$

where the QoS constraints (1b) and (1c) ensure that the total latency and energy consumption meet their respective target

maximum values for ML tasks. In the RAN slicing problem, instead, the ES aims to maximize the minimum inference accuracy across all MDs under latency and energy constraints, by jointly optimizing the bandwidth and computing resource allocation and channel coding rate. The RAN slicing problem (P2) can then be formulated as:

$$\mathrm{P2}: \quad \max_{\boldsymbol{\phi},\boldsymbol{b},\boldsymbol{f}^e} \frac{1}{T}\sum_{0\le t\le T}\min_{m\in\mathcal{M}}\xi_m(t) \tag{2a}$$

$$s.t \quad \tau_m^s(t) \le \tau_m^{\mathrm{th}}(t), \forall m\in\mathcal{M}, 0\le t\le T, \tag{2b}$$

$$E_m^s(t) \le E_m^{\mathrm{th}}(t), \forall m\in\mathcal{M}, 0\le t\le T, \tag{2c}$$

$$\sum_{m\in\mathcal{M}} b_m(t) \le B, 0\le t\le T, \tag{2d}$$

$$\sum_{m\in\mathcal{M}} f_m^e(t) \le F^e, 0\le t\le T \tag{2e}$$

Constraints (2d)–(2e) impose resource limits on the total available bandwidth B and computing capacity F^e across all MDs.

It can be seen that P1 is a nonlinear integer programming (NLIP) problem with discrete variables ($\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$), while P2 is a mixed-integer nonlinear programming (MINLP) problem that involves intricate coupling among the discrete variables ($\boldsymbol{\phi}$) and the continuous variables ($\boldsymbol{b}, \boldsymbol{f}^e$). Both P1 and P2 are NP-hard. This can be proved by constructing a polynomial-time reduction from the generalized assignment problem (GAP)—a problem already known to be NP-hard.

Also, and most importantly, due to the lack of an analytical expression for the optimization objective in P1 and P2 (i.e., inference accuracy), conventional convex and non-convex optimization methods cannot be directly applied. Instead, as detailed next, we propose a distributed online optimization framework to iteratively solve P1 and P2.

A. Gate Unit Control on MDs

In time slot t, every MD m has to solve the following problem:

$$\mathrm{P1}': \quad \max_{\beta_m(t),\lambda_m(t)} \xi_m(t) \quad s.t \quad (1b), (1c). \tag{3}$$

In P1′, an MD faces two uncertainties. It cannot predict (i) the resource allocation and channel coding rate chosen by the ES, impacting its accuracy, latency, and energy consumption, and (ii) the bit errors and packet loss due to the wireless channel, affecting feature integrity and making inference accuracy difficult to estimate. This decision-making process can be modeled as a multi-agent contextual MAB problem with the following properties: *(i)* finite, discrete configurations for MDs, *(ii)* the MDs' actions impact only the benefits they can get in the current time slot, and *(iii)* the MDs can receive feedback to refine strategies in the current time slot. Next, we define the context, action, and reward space for such a problem.

(i) Context. Let $\boldsymbol{s}_m(t)\in\mathcal{S}$ be MD m's context at time t, including task requirements and estimated achievable rate $\boldsymbol{s}_m(t)=\{\tau_m^{th}(t), E_m^{th}(t), \omega_m^{\mathrm{t}}(t), \omega_m^e(t), \hat{\delta}_m(t)\}$. MDs estimate the CSI via uplink pilots and ES's feedback, to obtain $\hat{\delta}_m(t)$.

(ii) Arms. Each arm corresponds to a branch of the dynamic split DNN model. The arm $\boldsymbol{a}_m(t)\in\mathcal{A}$ chosen by MD m at time t is $\boldsymbol{a}_m(t)=(\beta_m(t), \lambda_m(t))$, with $|\mathcal{A}|=|\mathcal{B}|\times|\Lambda|$.

(iii) Rewards. After resource slicing and ML task execution, the MDs receive from the ES: the accuracy $\xi_m(t)$, transmission latency $\tau_m^{\mathrm{t}}(t)$, edge processing latency $\tau_m^e(t)$, and transmission energy consumption $E_m^{\mathrm{t}}(t)$. Latency and energy consumption targets are treated as soft constraints. By denoting with $\mathrm{erf}(\cdot)$ the Gaussian error function, the reward $r_m(t)$ is given by:

$$\begin{aligned} r_m(t) =& \xi_m(t) + \omega_m^{\mathrm{t}}(t)\,\mathrm{erf}\left(\tau_m^{th}(t)-\tau_m^s(t)\right) \\ &+ \omega_m^e(t)\,\mathrm{erf}\left(E_m^{th}(t)-E_m^s(t)\right). \end{aligned} \tag{4}$$

The reward function in P1′ is non-linear, with rewards for different gate unit control decisions being correlated. Small changes in $\beta_m(t)$ or $\lambda_m(t)$ lead to gradual shifts in system latency, energy consumption, and accuracy. This correlation allows us to infer unobserved context-action pairs, reducing exploration overhead. As MDs must make decisions without knowing the ES's strategy, prior knowledge of action-reward likelihoods is beneficial. We adopt the Gaussian process-based upper-confidence-bound (GP-UCB) algorithm, modeling the reward function as samples from a Gaussian process (GP) over the context-action space. The upper confidence bound (UCB) algorithm is used to select the best action. Let $z\in\mathcal{Z}=\mathcal{S}\times\mathcal{A}$ denote a context-action pair, and $z_m(t)$ the context-action pair of MD m at time slot t. Let $GP(\mu_m(z), k_m(z,z'))$ be the GP function estimator for the reward function, where $\mu_m(z)$ is the mean function and $k_m(z,z')$ is the covariance function or the kernel of MD m. Assume a zero mean prior with bounded variance less than 1. Let $\hat{\boldsymbol{r}}_m(t)=\{\hat{r}_m(1),\ldots,\hat{r}_m(t)\}$ denote the noisy reward sample vector, with Gaussian noise $N(0,(\sigma^n)^2)$. In time slot t, MD m updates the posterior distribution of the reward function GP by:

$$\mu_m(z) \leftarrow \boldsymbol{k}_m\left(z, z_m(t)\right)^\top \left(\boldsymbol{K}_m + (\sigma^n)^2 I\right)^{-1} \hat{\boldsymbol{r}}_m(t), \tag{5}$$

$$\begin{aligned} k_m(z,z') \leftarrow & k_m(z,z') - \boldsymbol{k}_m\left(z, z_m(t)\right) \\ & \times \left(\boldsymbol{K}_m\left(z_m(t)\right) + (\sigma^n)^2 I\right)^{-1} \boldsymbol{k}_m\left(z, z_m'(t)\right), \end{aligned} \tag{6}$$

where $\boldsymbol{k}_m\left(z, z_m(t)\right) = \left[k_m\left(z, z_m(1)\right), \ldots, k_m\left(z, z_m(t)\right)\right]^\top$ and we use $\mathbf{K}_m\left(z_m(t)\right) = \left[k_m\left(z,z'\right)\right]_{z,z'\in\mathcal{Z}_m(t)}$ to denote the covariance vector and the kernel matrix. Based on the posterior distribution, the MDs can estimate the unobserved values of $z\in\mathcal{Z}$. Similarly to [21], we adopt a kernel satisfying stationarity and anisotropy. The distance between two context-action pair z and z' is given by $\sqrt{(z-z')^\top \boldsymbol{L}^{-2}(z-z')}$, where $\boldsymbol{L}=\mathrm{diag}(\boldsymbol{l})$ is a diagonal matrix, and $\boldsymbol{l}$is the length-scale vector. Using the anisotropic Matérn kernel [22], the kernel function for the reward function GP of MD m is $k_m\left(z,z'\right) = \left(1+\sqrt{3(z-z')^\top \boldsymbol{L}^{-2}(z-z')}\right) \times \exp\left(-\sqrt{3(z-z')^\top \boldsymbol{L}^{-2}(z-z')}\right)$. Given the context-action set, MD m leverages the UCB policy as the acquisition function to select the dynamic DNN parameters as:

$$\boldsymbol{a}_m(t) = \arg\max_{\boldsymbol{a}\in\mathcal{A}} \left[\mu_m(z) + \sqrt{\omega_m^a(t)} k_m(z,z')\right], \tag{7}$$

where $\omega_m^a(t)$ is a dynamic weight that controls the trade-off between exploration and exploitation. Specifically, we select $\omega_m^a(t)$ based on an upper bound of the Reproductive Kernel Hilbert Space (RKHS) norm and the maximum mutual information gain from past observations (see [22] for the specific formula). To evaluate the performance of different acquisition functions, the UCB-based acquisition function can be replaced with Thompson sampling (TS) [23]. Instead of selecting actions based on the upper confidence bound in (7), TS samples from the posterior distribution of the reward function and selects the action that maximizes the sampled value. Notably, the TS-based acquisition function can be written as: $\boldsymbol{a}_m(t)=\arg\max_{\boldsymbol{a}\in\mathcal{A}}\tilde{\mu}_m(z)$, where $\tilde{\mu}_m(z)$ is a sample drawn from the posterior distribution of the reward function. TS can be considered as another deployment version of OMNIS.

Next, we analyze the regret performance of the multi-agent GP-UCB algorithm. Let the optimal and actual expected rewards in time slot t for MD m be $\max_{\boldsymbol{a}\in\mathcal{A}}\mathbb{E}[r_m(t)|\boldsymbol{s}_m(t)]$ and $\mathbb{E}[r_m(t)|\boldsymbol{s}_m(t),\boldsymbol{a}_m(t)]$, respectively. The regret for MD m is $R_m(t)=\max_{\boldsymbol{a}\in\mathcal{A}}\mathbb{E}[r_m(t)|\boldsymbol{s}_m(t)]-\mathbb{E}[r_m(t)|\boldsymbol{s}_m(t),\boldsymbol{a}_m(t)]$. The cumulative regret over T time slots for MD m is $R_m^T=\sum_{t=1}^{T}R_m(t)$, while the system's cumulative regret is $R^T=\sum_{m\in\mathcal{M}}R_m(T)$. Theorem 1 provides the expression for R^T when using the GP-UCB algorithm.

Theorem 1. *The system cumulative regret R^T for gate unit control can be obtained for any $T\geq 1$ as:*

$$P\left(R^{\mathrm{t}}\leq M\left(\sqrt{\frac{8T\omega^a(T)\gamma^T}{\log\left(1+(\sigma^n)^{-2}\right)}}+2\right)\right)\geq 1-M\eta,\quad,\qquad(8)$$

where $\eta\in(0,1)$, $\gamma^T=\mathcal{O}\left(\tau^{\frac{\dim(\mathbf{z})(\dim(\mathbf{z})+1)}{3+\dim(\mathbf{z})(\dim(\mathbf{z})+1)}}\log T\right)$, $\dim(z)$ *is the dimension of* z *that satisfies* $\dim(z)=\dim\left(\boldsymbol{a}_m(t)\right)+\dim\left(\boldsymbol{s}_m(t)\right)$.

Proof. Using the Matérn kernel, the individual regret bound for MD m is given in [21], [22], as:

$$P\left(R_m^{\mathrm{t}}\leq\sqrt{\frac{8T\omega^a(T)\gamma^T}{\log\left(1+(\sigma^n)^{-2}\right)}}+2\right)\geq 1-\eta,\quad\forall T\geq 1.$$

Since each MD operates independently under the same regret bound, using the Boole's inequality (union bound), the probability that at least one MD violates this bound is at most $M\cdot\eta$. Summing over all MDs, we get the regret bound as in (8). □

B. RAN Resource Slicing on ES

Before MD m transmits the head model output to the ES, it first sends the task requirements $\{\tau_m^{th}(t),E_m^{th}(t),\omega_m^{\mathrm{t}}(t),\omega_m^e(t)\}$, the gate unit control results $\boldsymbol{a}_m(t)$, the local processing latency $\tau_m^l(t)$, and the energy consumption $E_m^l(t)$ to the ES for solving P2. Upon receiving this data from the MDs, in every time slot t the ES solves the following problem:

$$\mathrm{P2}':\max_{\boldsymbol{b}(t),\boldsymbol{f}^e(t),\boldsymbol{\phi}(t)}\min_{m\in\mathcal{M}}\xi_m(t)\quad s.t\quad(2b),(2c),(2d),(2e).\qquad(9)$$

The coupling between continuous $(\boldsymbol{b},\boldsymbol{f}^e)$ and discrete $(\boldsymbol{\phi})$ variables still makes the problem intractable. Thus, using the BCD method, we further decompose P2′ into three sub-problems: *(i)* bandwidth allocation, *(ii)* GPU frequency allocation, and *(iii)* channel coding rate selection. Each sub-problem is solved alternately by fixing the other two variables until convergence, i.e., the stabilization of decision variables and the overall objective value across iterations, or the algorithm reaching a predefined maximum number of iterations. Notably, the closed-form solution of sub-problem *(i)* can be obtained using the Lagrange multiplier method; sub-problem *(ii)* can be efficiently solved using off-the-shelf convex optimization tools such as CVXPY; sub-problem *(iii)* can solved by exhaustive search. This decomposition enables scalable optimization even under complex coupling between variables.

Note that the ES's resource allocation does not affect the inference accuracy of MDs when other variables are fixed. The ES's objective is to minimize QoS soft constraint violations (2b) and (2c), while satisfying resource capacity hard constraints (2d)—(2e), given the gate unit control strategy $\boldsymbol{a}_m(t)$ and initial channel coding rate $\boldsymbol{\phi}^{[0]}(t)$. By incorporating the QoS constraints and removing the irrelevant term (inference accuracy $\xi_m(t)$), the bandwidth allocation problem in time slot t becomes:

$$\mathrm{P2.1}':\min_{\boldsymbol{b}(t)}\sum_{m\in\mathcal{M}}\frac{d_m(t)\left(\omega_m^{\mathrm{t}}+\omega_m^e p_m(t)\right)}{b_m(t)\log\det\left(\boldsymbol{I}_{\rho^e}+\frac{\boldsymbol{H}_m(t)\boldsymbol{H}_m^{\mathrm{H}}(t)}{(\sigma^n)^2}\right)}\qquad(10a)$$

$$s.t\quad(2d),0\leq b_m(t)\leq B,b_m(t)\in\mathbb{R},\forall m\in\mathcal{M}.\qquad(10b)$$

Let $d'_m(t)=\frac{d_m(t)\left(\omega_m^{\mathrm{t}}+\omega_m^e p_m(t)\right)}{\log\det\left(\boldsymbol{I}_{\rho^e}+\frac{\boldsymbol{H}_m(t)\boldsymbol{H}_m^{\mathrm{H}}(t)}{(\sigma^n)^2}\right)}$. We build a Lagrange function $\mathcal{L}(\boldsymbol{b},\iota^b)$ by incorporating constraint (2d) into (10a):

$$\mathcal{L}(\boldsymbol{b},\iota^b)=\sum_{m\in\mathcal{M}}\frac{d'_m(t)}{b_m(t)}+\iota^b\left(\sum_{m\in\mathcal{M}}b_m(t)-B\right),\qquad(11)$$

where ι^b is a Lagrange multiplier. Taking the partial derivative of $b_m(t)$ and setting $\mathcal{L}(\boldsymbol{b},\iota^b)=0$, we have $b_m^*(t)=\sqrt{d'_m(t)/\iota^b}$. Based on constraint (2d), we have $\sum_{m\in\mathcal{M}}\frac{d'_m}{\iota^b}=B$. Then we can obtain $\iota^b=\frac{1}{B}\sum_{m\in\mathcal{M}}\sqrt{d'_m(t)}$, thus the optimal bandwidth allocation strategy is $b_m^*(t)=B\sqrt{d'_m(t)}/\sum_{m\in\mathcal{M}}\sqrt{d'_m(t)}$.

Similarly, the computing resource allocation problem in time slot t can be written as:

$$\mathrm{P2.2}':\min_{\boldsymbol{f}^e(t)}\sum_{m\in\mathcal{M}}\left(\frac{\omega_m^{\mathrm{t}}\psi_m^{\mathrm{t}}(t)}{f_m^e(t)c^e\chi^e}+\frac{\omega_m^e\zeta^e\left(f_m^e(t)\right)^2\psi_m^{\mathrm{t}}(t)}{c^e\chi^e}\right),\qquad(12a)$$

$$s.t\quad(2e),0\leq f_m(t)\leq F^e,c_m(t)\in\mathbb{R},\forall m\in\mathcal{M}.\qquad(12b)$$

The first term in the above sum is convex with respect to $f_m^e(t)$ since it has the form of an inverse function, which is convex for positive values. The second term is also convex, as it is a quadratic function of $f_m^e(t)$. Since the sum of convex functions is still convex, so is the objective function. Further, the constraint (2e) is linear, thus the feasible set is convex.

Hence, P2.2′ is convex and can be efficiently solved using solvers such as CVXPY.

Given the gate unit control strategy and the resource allocation strategy, the ES then has to solve the following channel coding rate selection problem:

$$\text{P2.3}': \quad \max_{\phi_m(t)} \min_{m\in\mathcal{M}} \xi_m(t) \quad s.t \quad (2b), (2c). \tag{13}$$

Importantly, since the other variables are fixed, the channel coding rate selected by the ES for MD m does not affect the accuracy, latency, or energy consumption of other MDs. Thus, the ES can independently determine each MD's channel coding rate, and an exhaustive search over all possible coding rates can be performed. The ES aims to select the coding rate that maximizes inference accuracy while satisfying the QoS constraints (2b) and (2c). If no rate satisfies the constraints, the ES chooses the one minimizing the penalty term in (4). To determine inference accuracy for a selected rate, the ES can use pre-computed accuracy curves under different SNR and payload conditions to guide decision-making. The overall computing complexity to solve P1 and P2 by using the proposed optimization framework is given in Theorem 2.

Theorem 2. *Assuming that* P2′ *requires* τ^o *iterations to converge, the total computing complexity of solving* P1 *and* P2 *for* T *time slots is* $\mathcal{O}(M(T^4+T|\mathcal{A}|+\tau^o(T+|\Phi|+M^2T)))$.

Proof. In slot t, the GP-UCB algorithm updates the GP model for each MD m, involving matrix inversions and kernel computations with complexity $\mathcal{O}(t^3)$ [22]. Computing the UCB values adds $\mathcal{O}(t)$, and action selection over a discrete space $\mathcal{A}$ contributes $\mathcal{O}(|\mathcal{A}|)$. With M MDs, the total complexity of the multi-agent GP-UCB algorithm is thus $\mathcal{O}(Mt^3+M|\mathcal{A}|)$. Meanwhile, the ES allocates bandwidth resources by solving P2.1′ with complexity $\mathcal{O}(M)$. Using Interior-Point Method in CVXPY, P2.2′ becomes a second-order cone programming (SOCP), whose complexity is $\mathcal{O}(M^3)$. Since the channel coding rate can be selected independently for each MD, the complexity of solving P2.3′ is $\mathcal{O}(M|\Phi|)$. Summing over T time slots $\sum_{t=1}^{T} t^3=\mathcal{O}(T^4)$, the total complexity is $\mathcal{O}(MT^4+MT|\mathcal{A}|+MT\tau^o+M\tau^o|\Phi|+M^3T\tau^o)=\mathcal{O}(M(T^4+T|\mathcal{A}|+\tau^o(T+|\Phi|+M^2T)))$. Although the complexity expression involves T^4, our evaluation shows that the proposed algorithm converges swiftly. □

VI. Performance Evaluation

Experimental Settings. We consider an ES-centered cell of radius 250,m, where MDs are uniformly and randomly deployed. Each MD's GPU frequency, number of cores, and FLOPs per cycle are independently drawn from $[1.2, 2]$,GHz, $[512, 1024]$, and $[8, 16]$, respectively; the ES parameters are drawn from $[3, 5]$,GHz, $[8192, 16384]$, and $[16, 32]$. The MDs and ES are equipped with N_m=4 and N_e=64 antennas, and the system bandwidth is 100,kHz. At each slot t, the latency and energy thresholds are sampled as $\tau_m^{th}(t) \sim \mathcal{U}(0.8, 1.5)$,s and $E_m^{th}(t) \sim \mathcal{U}(0.8, 1.5)$,J, while the latency weight is $\omega_m^t(t) \sim 1.5,\mathcal{U}(0, 1)$ with $\omega_m^e(t) = 1 - \omega_m^t(t)$. Mobility follows the random point model with random initial placement and speed $v \sim \mathcal{U}(10, 20)$,m/s. The MIMO channel includes large-scale fading under the log-distance pathloss model $128.1 + 37.6\log_{10}(d)$,dB and small-scale Rayleigh fading; AWGN power is modeled as $\mathcal{N}(0, -174+10\log_{10}(B))$,dBm. We use BPSK modulation and LDPC coding with rate $\phi \in 1, 2/3, 5/6$, and set the channel estimation error variance to σ^h=0.5.

Each MD performs semantic segmentation on one image per slot using the COCO dataset [5]. Performance is measured by COCO mAP@[0.5:0.95] (IoU thresholds from 0.50 to 0.95 in steps of 0.05). The dynamic split DNN is built on Mask R-CNN with a ResNet-50 backbone. The head encoder comprises four convolutional layers: the first mirrors the ResNet-50 stem (BN–ReLU–MaxPool) and the next two apply stride-2 convolutions for downsampling. For the tail, we set the split point j at the output of ResNet-50's first block, and apply reconstruction loss to the outputs of the remaining three blocks; the decoder uses two deconvolutional and two convolutional layers. GDN is used for rANS renormalization, while Box quantization follows a standard bottleneck design. Given the error sensitivity of entropy quantization, we evaluate only standard and Box quantization with three channel settings (3, 6, and 12). Finally, we distill from a Mask R-CNN teacher (mAP 41.80%) and train the head/tail models using the two-stage procedure in [10].

Benchmarks. We compare OMNIS against three benchmarks, modified or extended from state-of-the-art RAN slicing frameworks for fair comparison. Notice that our scheme is labeled as **"OMNIS-UCB"** or **"OMNIS-TS"**, depending on whether UCB or TS is used as action acquisition function.

- *Greedy-based Optimization (GDO)* [2], [8]: SEM-O-RAN [8] is modified by replacing its "data compression scaling factor" with the compression mechanism of our dynamic DNN model. The ES employs a greedy algorithm to select dynamic quantization parameters. Initially, the ES randomly assigns neural network parameters to each MD and solves P2′. It then reallocates models, assigning less (more) compressed models to MDs with the lowest (highest) accuracy, iteratively solving P2′ until no further improvement in the objective is possible.
- *Resource Slicing with Static Models (RSS)* [24]: RSS employs a static version of the proposed inference model. Specifically, each MD's quantization parameters and channel coding rate remain fixed across all time slots, thus requiring only the ES to solve P2′ in each slot.
- *Centralized Optimization (CTO)* [21]: CTO implements a centralized contextual MAB agent at the ES. In each slot, the ES employs the GP-UCB and BCD algorithms to iteratively solve P1′ and P2′. The objective of P1′ is modified to align with P2′ for centralized optimization.

We evaluate all schemes under different parameter settings in terms of reward, latency, energy consumption, inference accuracy, QoS violation probability, and violation excess (i.e., the amount by which QoS constraints are exceeded). All results are averaged over 3 independent runs. Fig. 5 reports the performance averaged over 8 MDs and 200 time slots. OM-

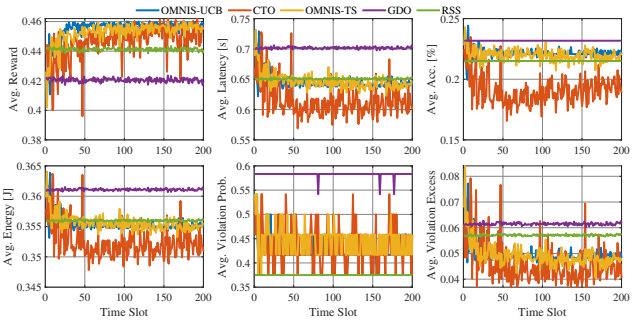

Fig. 5: Convergence of different schemes: average reward (top-left), latency (top-middle), accuracy (top-right), energy consumption (bottom-left), violation probability (bottom-middle), and average violation excess (bottom-right).

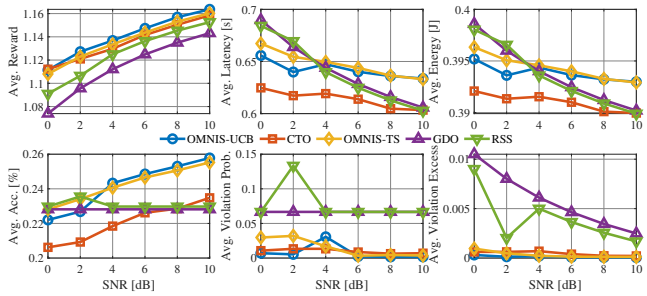

Fig. 6: Average performance for varying SNR values.

NIS (including OMNIS-TS and OMNIS-UCB, which behave similarly) converges rapidly: reward and accuracy stabilize within 30 slots, while latency and energy also reach steady states. OMNIS attains the highest accuracy and the best overall reward, with near-optimal latency and energy, and it substantially reduces QoS violations and over-expenditure, yielding more stable performance over time. Although centralized optimization (CTO) eventually approaches similar performance, it incurs about 13× higher runtime. In contrast, RSS and GDO do not adapt well to dynamic contexts, resulting in consistently suboptimal outcomes. Fig. 6 shows average performance versus SNR for 5 MDs over 200 slots. As SNR increases, higher transmission rates and lower BER reduce latency and energy and improve accuracy; OMNIS consistently achieves the highest reward and accuracy. While its latency and energy are slightly higher at high SNR, OMNIS maintains a low QoS violation probability and efficiently operates within constraints. Compared with RSS, OMNIS-UCB improves accuracy by up to 11.92% and reduces QoS violation probability and excess by up to 11.3× and 15.2×, respectively. Fig. 7 varies the number of MDs. As contention at the ES intensifies, delay increases and average reward decreases. Transmission energy grows with delay, whereas the dominant computing energy (proportional to the square of GPU frequency) decreases due to frequency sharing. When the number of MDs exceeds six, QoS violations rise sharply, leading to the selection of higher-compression branches and a corresponding accuracy drop. Nevertheless, OMNIS scales well and achieves the best reward, highest accuracy, and the

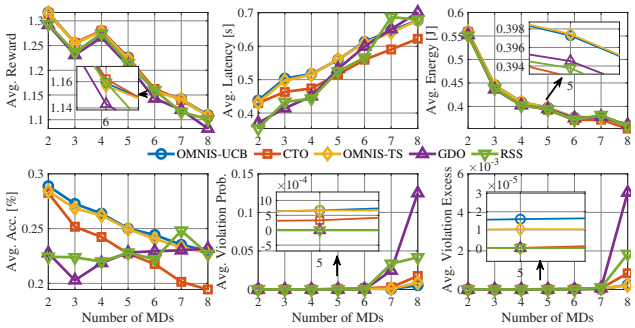

Fig. 7: Average performance for different MD numbers.

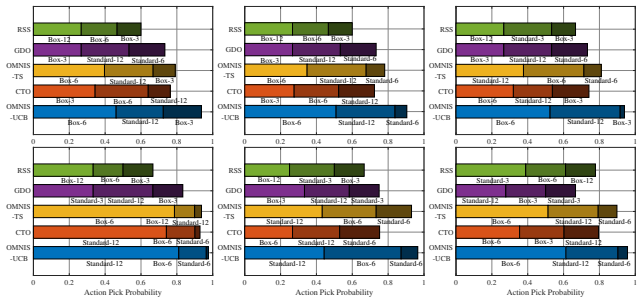

Fig. 8: Model pick probability for SNR=2 dB (top-left), 4 dB (top-middle), 6 dB (top-right), and 2 (bottom-left), 4 (bottom-middle), 6 (bottom-right) MDs.

fewest QoS violations even beyond six MDs; relative to RSS, OMNIS-UCB improves accuracy by up to 22.85% and reduces QoS violation probability and excess by up to 12.5× and 2.9×, respectively.

Fig. 8 shows the top-3 most selected models under varying SNRs and MD counts. As SNR increases or MDs decrease, OMNIS selects Standard-12 more frequently for its higher payload and accuracy. Better channel conditions allow larger payloads within QoS constraints, enabling use of low-compression models. Under poor SNR or high MD density, OMNIS adapts by selecting more compressed models (e.g., Box-6), balancing inference performance and QoS constraints. This demonstrates OMNIS's capability to dynamically select optimal models based on system conditions.

VII. Conclusions

We proposed OMNIS, a semantic slicing framework for edge computing. OMNIS overcomes the limitations of static slicing by leveraging dynamic split neural networks for adaptive quantization and bit error resilience. Notably, we introduced "Box quantization" to ensure efficient and reliable task offloading. To optimize the system performance, we formulated two optimization problems for MDs and the ES. The former maximizes inference accuracy under latency and energy constraints using a multi-agent contextual MAB algorithm. The latter maximizes the fairness of inference accuracy via the BCD method with convex optimization. Our results show that OMNIS improves inference accuracy by up to 22.85% while significantly reducing the QoS constraint violation probability by up to 10x.

REFERENCES

[1] K. Huang and W. Gao, "Real-time neural network inference on extremely weak devices: agile offloading with explainable ai," in *ACM MobiCom*, (New York, NY, USA), p. 200–213, ACM, 2022.

[2] X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen, "Joint multiuser dnn partitioning and computational resource allocation for collaborative edge intelligence," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9511–9522, 2021.

[3] W. Wu, N. Chen, C. Zhou, M. Li, X. Shen, W. Zhuang, and X. Li, "Dynamic ran slicing for service-oriented vehicular networks via constrained learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2076–2089, 2021.

[4] C. Puligheddu, N. Varshney, T. Hassan, J. Ashdown, F. Restuccia, and C. F. Chiasserini, "Offloadnn: Shaping dnns for scalable offloading of computer vision tasks at the edge," in *IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, pp. 624–634, 2024.

[5] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[6] K. He *et al.*, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[7] Z. Ji and Z. Qin, "Computational offloading in semantic-aware cloud-edge-end collaborative networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 18, no. 7, pp. 1235–1248, 2024.

[8] C. Puligheddu, J. Ashdown, C. F. Chiasserini, and F. Restuccia, "Sem-o-ran: Semantic o-ran slicing for mobile edge offloading of computer vision tasks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 7, pp. 7785–7800, 2024.

[9] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Comput. Surv.*, vol. 55, Dec. 2022.

[10] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato, and F. Restuccia, "Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing," in *IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 337–346, 2022.

[11] G. Chen, Q. Wu, R. Liu, J. Wu, and C. Fang, "Irs aided mec systems with binary offloading: A unified framework for dynamic irs beamforming," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 2, pp. 349–365, 2023.

[12] L. Zhao, Z. Zhao, E. Zhang, A. Hawbani, A. Y. Al-Dubai, Z. Tan, and A. Hussain, "A digital twin-assisted intelligent partial offloading approach for vehicular edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 11, pp. 3386–3400, 2023.

[13] L. Qin, H. Lu, Y. Chen, B. Chong, and F. Wu, "Toward decentralized task offloading and resource allocation in user-centric mec," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 11807–11823, 2024.

[14] H. Xiang, S. Yan, and M. Peng, "A realization of fog-ran slicing via deep reinforcement learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2515–2527, 2020.

[15] S. D'Oro, F. Restuccia, T. Melodia, and S. Palazzo, "Low-complexity distributed radio access network slicing: Algorithms and experimental results," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2815–2828, 2018.

[16] Y. Sun, S. Qin, G. Feng, L. Zhang, and M. A. Imran, "Service provisioning framework for ran slicing: User admissibility, slice association and bandwidth allocation," *IEEE Transactions on Mobile Computing*, vol. 20, no. 12, pp. 3409–3422, 2021.

[17] Q. Fu, H. Xie, Z. Qin, G. Slabaugh, and X. Tao, "Vector quantized semantic communication system," *IEEE Wireless Communications Letters*, vol. 12, no. 6, pp. 982–986, 2023.

[18] D. Huang, F. Gao, X. Tao, Q. Du, and J. Lu, "Toward semantic communications: Deep learning-based image semantic coding," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 1, pp. 55–71, 2023.

[19] J. Shao, Y. Mao, and J. Zhang, "Learning task-oriented communication for edge inference: An information bottleneck approach," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 197–211, 2022.

[20] J. Duda, "Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding," *arXiv:1311.2540*, 11 2013.

[21] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Perez, and G. Iosifidis, "Bayesian online learning for energy-aware resource orchestration in virtualized rans," in *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1–10, 2021.

[22] A. Krause and C. S. Ong, "Contextual gaussian process bandit optimization," in *25th International Conference on Neural Information Processing Systems*, NIPS'11, (Red Hook, NY, USA), Curran Associates Inc., 2011.

[23] T. Ouyang, X. Chen, Z. Zhou, R. Li, and X. Tang, "Adaptive user-managed service placement for mobile edge computing via contextual multi-armed bandit learning," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, 2023.

[24] J. Yan, Q. Lu, and G. B. Giannakis, "Bayesian optimization for online management in dynamic mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 23, no. 4, pp. 3425–3436, 2024.