Cooperative Multi-Agent Deep Reinforcement Learning for Dynamic Task Execution and Resource Allocation in Vehicular Edge Computing

Robert Rauch, Zdenek Becvar, *Senior Member, IEEE*, Pavel Mach, *Member, IEEE*, Juraj Gazda

***Abstract*—Computer vision plays a crucial role in enabling connected autonomous vehicles (CAVs) to observe and comprehend their surroundings. The computer vision tasks are typically based on convolutional neural networks (CNNs). However, CNNs often require significant processing power. Techniques like early exiting and split computing enhance CNN task execution latency and adaptability to varying environmental conditions. Since the split computing introduces additional overhead for offloading of the task from the CAV to an edge servers, we incorporate multiple autoencoders within each split point to enhance the adaptability of splitting under varying environmental conditions. However, the autoencoders introduce an additional layer of complexity related to the selection of the optimal compression strategy alongside the splitting and exiting decisions. To tackle this challenge, we introduce a novel approach based on the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm. This algorithm dynamically and jointly determines the most suitable exit point, split point, and autoencoder. Furthermore, the MADDPG-based approach considers other CAVs when selecting action, promoting cooperation among CAVs. Our results demonstrate that the proposed approach reduces latency up to 44.4% while maintaining at least comparable or even higher accuracy of the computed vision outcome compared to the state-of-the-art solutions.**

***Index Terms*—Autonomous Mobility, Computer Vision, Cooperative Multi-Agent, Deep Learning, Dynamic Task Execution**

This work was supported by The Slovak Research and Development Agency project no. APVV SK-CZ-RD-21-0028, APVV-23-0512, the Slovak Academy of Sciences project no. VEGA 1/0685/23, and by the Ministry of Education, Youth and Sports, Czech Republic, under the project LUASK22064.

Part of the Research results was obtained using the computational resources procured in the national project National competence centre for high performance computing (project code: 311070AKF2) funded by European Regional Development Fund, EU Structural Funds Informatization of society, Operational Program Integrated Infrastructure.

Icons used in this work were made by Freepik, vectorsmarket15, rcherem, Vectors Market and Ylivdesign from www.flaticon.com.

During the preparation of this work the authors used AI tool in order to improve language and readability. The ideas and content remain the sole responsibility of the authors.

R. Rauch and J. Gazda are with the Department of Computers and Informatics, Technical University of Kosice, 04200 Kosice, Slovakia. (e-mail: robert.rauch@tuke.sk; juraj.gazda@tuke.sk)

Z. Becvar and P. Mach are with the Deparment of Telecommunication Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic. (e-mail: zdenek.becvar@fel.cvut.cz; machp2@fel.cvut.cz)

I. Introduction

The autonomous driving has a potential to transform the transportation by enabling connected autonomous vehicles (CAVs) to operate without human intervention [1]. A key challenge in the autonomous driving is, however, to ensure the real-time responsiveness of all systems in a highly dynamic and ever-changing environment. The CAVs usually generate and process huge amount of high-dimensional data from various sensors [2], [3], allowing the CAVs to collect data related to their inside as well as outside environment.

The sensor data related to the environment sensing are commonly processed using various computer vision techniques [4]. Deep learning techniques, predominantly based on convolutional neural networks (CNNs), are adopted for the computer vision tasks, such as image classification [5], image segmentation [6], or object detection [7]. All these computer vision tasks for vehicular applications require an execution under a latency constraint and, at the same time, a high accuracy of the execution results should be also ensured. Unfortunately, the processing of the computer vision tasks with strict latency constraint directly on the CAV may not be possible due to limited computational capabilities of the CAVs [8]. Hence, the processing of the computer vision tasks can be offloaded to edge computing servers with relatively high computational capabilities. This offloading concept, where the CAVs can delegate computationally demanding tasks to the computing servers in their vicinity, is known as the Vehicular Edge Computing (VEC) [9].

The key challenge related to the offloading of computation tasks from the CAV to the edge server is an allocation of communication and computation resources. To this end, various approaches, including heuristic [10], [11] and machine or deep learning-based [12], [13], [14], are employed. However, none of these studies specifically address the offloading of the computer vision tasks, which are crucial for the autonomous driving. The computer vision tasks bring number of additional challenges, such as the need for sequential execution [15], high data volume generation [2], inherent variability and complexity [16], and requirement on low latency response [2]. These challenges combined all together potentially strain communication and computational resources for VEC [2].

To reduce both communication and computation load imposed by the computer vision tasks, the task partitioning can be adopted, see, e.g., [17], [18]. As a result, each task can be divided into several subtasks, some executed directly on

the CAV while the rest offloaded to the edge server(s). The sequential processing of CNN can be split between two places [19] to leverage the computational capabilities of both the CAV and the edge server to process CNN. In such case, an initial part of CNN up to a certain layer, denoted as a split point, is processed on the CAV. Then, the intermediate data from the last layer of CNN processed at the CAV is transferred to the edge server, where the processing of the remaining CNN layers continues from the split point to the output layer. The split of CNN's processing enables to reduce the load of communication as well as computing resources of the edge server compared to the offloading of the whole processing of CNN to the edge server [19], [20], [21].

The optimization of split computing is addressed in [22], where multi-armed bandit approach is employed to find the split point leading to the lowest latency of execution. However, the most prominent optimal strategies for task offloading involve either transferring the entire task to the edge server or executing the entire task on the CAV [19]. This is often favored due to the significant increase in communication latency associated with the extensive data generated by convolutional layers, that would have to be offloaded.

Transmitting a huge amount of data generated by the computer vision applications in the CAVs to the edge server is a challenge in scenarios with a high number of CAVs (e.g., in downtown during busy hours) or in the areas with limited availability and/or insufficient quality of wireless resources between the CAV(s) and the edge server(s) (e.g., in rural areas). This challenge can be partially suppressed by using autoencoders to compress data in the CAV and decompress on the server [23]. The level and complexity of the compression rate determines the loss in accuracy and the amount of data to be transferred [24]. The performance of autoencoders for various layers or compression ratios on a given layer is investigated, for example, in [23], [24]. However, these studies do not address the problem of dynamically selecting an appropriate autoencoder depending on the environmental conditions (e.g., wireless channel quality and availability of radio resources). Besides, even if both the CNN splitting and the autoencoders may improve the overall CNN-based task execution latency for the CAVs, it may still not be sufficient for delay sensitive applications related to autonomous driving [25].

To further optimize the performance of CNN, the CNN's architecture with early exits can also be adopted [26]. The early exits are understood as new branches in CNN [27]. These branches enable CNN to provide output (exit) at different points of the processing. On one hand, exiting through earlier branches can improve the latency of CNN. On the other hand, the early exit may compromise the accuracy of the CNN, since not all layers are exploited. Therefore, the selection of exit points allows to adjust the trade-off between the accuracy and the latency [28]. Besides, depending on a technique adopted for the early exiting, the early exit may also infer an additional computational overhead, when assessing the confidence of each exit's output [27]. Some techniques, however, decide exit before execution and, thus, may not infer computational overhead. The technique proposed in [25] determines the exit based on current environmental conditions (e.g., wireless channel quality and availability of radio resources), while the approach presented in [29] decides the exit by incorporating the network's confidence for exiting prior to each early exit. These approaches allows CNN to identify the appropriate exit prior to its selection.

Multiple techniques can be employed to reduce the task execution latency using both early exiting and split computing [30]. The work proposed in [25] suggests using a selective heuristic method. Furthermore, deep learning is adopted in [26] to find the optimal split and exit points. However, these approaches do not employ autoencoders or any other form of intermediate data compression. As [22] shows, this constrains the solution space significantly when determining the optimal strategy, which involves the selection of the most appropriate split point and exit point. Furthermore, both [25] and [26] approaches are greedy, which means they do not take other vehicles into account when selecting the optimal strategy. Consequently, in the scenarios with multiple CAVs, strict latency requirements, and high computational demands, the efficiency of the solutions described in [25] and [26] may be compromised.

Motivated by drawbacks and gaps of existing works, we target in this paper a scenario with the multiple CAVs running tasks related to the autonomous driving. These tasks can be processed locally in the CAV or (partially) offloaded to the edge computing server via shared communication resources. In this scenario, we minimize the execution latency of the task while maximizing the accuracy of the processing outcomes via joint selection of the CNNs' split point, exit point, and autoencoder. Moreover, unlike state-of-the-art works that adopt a greedy selection by each CAV, we encourage the CAVs adopting a cooperative behavior when choosing their strategy to effectively orchestrate both communication and edge computing resources among the CAVs.

The main contributions of this paper are summarized as follows:

- We formulate the problem of maximization of the CAVs' utility, represented by a weighted sum of the latency and accuracy of CNN outcomes, with constraints on guaranteeing the maximum latency of the overall task execution and the minimum required accuracy of the task processing. The solution is based on a joint selection of the CNNs' split point, exit point, and autoencoder. To the best of our knowledge no prior work incorporates a dynamic selection of the autoencoder together with the split and exit points selection.
- Given the very high complexity of the problem and very large search and observation spaces, we introduce a novel framework based on reinforcement learning. To this end, we first transform the problem into Markov Decision Process (MDP). Since the CAVs share communication and computing resources, any action of individual CAV influences other CAVs. To motivate cooperation among CAV, we adopt cooperative multi-agent techniques.
- Through simulations, we demonstrate that our proposed solution outperforms the existing state-of-the-art works in terms of reducing the task execution latency (by up to 44.4%), while maintaining comparable or even superior

accuracy of the outcomes of the computer vision task processing.

This paper is structured as follows. In Section II, we present the system model. In Section III, we formulate the problem as a non-linear programming and discuss its properties. In Section IV, we propose a novel algorithm to solve the problem. In Section V, we demonstrate the effectiveness of our algorithm through numerical simulations and comparisons with existing methods. In Section VI, we summarize our main contributions and suggest directions for future work.

II. System Model

This section first introduces a network model followed by CNN model representing the computation task. Then, we describe the model of the communication channel between the CAV and the edge server together with the computing model employed for CNN processing.

A. Network model

As in, e.g., [25], [26], we consider a set of CAVs $\mathcal{V}$ under coverage of a single edge server b collocated with a base station, see Fig. 1. We assume only single edge server as the novelty of our work is not related to migration and/or handover and tasks are offloaded in orders of a few milliseconds during which even fast moving vehicles cover very short distances. During the time interval t, the CAVs generate a set of computing tasks $\mathcal{Z}_{t,v} = \{z_{t,v,1}, z_{t,v,2}, ..., z_{t,v,|\mathcal{Z}_{t,v}|}\}$, where $z \in \mathcal{Z}_{t,v}$ represents a task and $|\mathcal{Z}_{t,v}|$ denotes cardinality of a set $\mathcal{Z}_{t,v}$. Each task $z \in \mathcal{Z}_{t,v}$ is associated to specific CAV v during time interval t and is characterized by its configuration (i.e., by its computational demand for CAV $I_{z,v}$, computational demand for edge server $I_{z,b}$, volume of the offloading data for given task $c^{\mathrm{UL}}_{z,l}$). Here, computational demand $I_{z,v}$ represents the computational resources the CAV must allocate to process the task locally, while computational demand $I_{z,b}$ quantifies the computational resources the edge server would require to process the task if it is offloaded. The tasks in $\mathcal{Z}_{t,v}$ are assumed to be generated periodically corresponding to, for example, a scenario with a camera sensor operating at a fixed frame rate. Note, that we assume all CAVs $\mathcal{V}$ to be generating set of tasks $\mathcal{Z}_{t,v}$ during each time interval t.

B. Convolutional Neural Network Model

In this section, we describe a CNN model as a task to be processed. In this regard, following [25], [26], we consider image classification tasks to be processed by CNN. In CAVs, image classification can be applied to various scenarios, such as lane detection [31], lane classification [31], road sign detection [32], or collision detection [33], among others.

Each task is modeled as CNN with a set of exits $E = \{e_1, e_2, ..., e_{|E|}\}$, where $|E|$ denotes the number of exits. Additionally, each CNN has a set of splits $S = \{s_1, s_2, ..., s_{|S|}\}$ and a corresponding set of autoencoders $A_s = \{a_{s,1}, a_{s,2}, ..., a_{s,|A_s|}\}$ for each split $s \in S$.

At each time interval t, the CAV dynamically updates the execution strategy of the CNN-based task according to the current network and environmental conditions (e.g., average latency of the task execution in the time interval t, average offloading time of all tasks in the time interval t) and task requirements (i.e., accuracy of the task processing and latency). The execution strategy consists of three components: the exit point, the split point, and the autoencoder. Thus, we formulate the execution strategy of the v-th CAV within the time interval t as:

$$\zeta_{t,v} = \{e_{t,v}, s_{t,v}, a_{s_{t,v},t,v}\}, \tag{1}$$

where $e_{t,v} \in E$, $s_{t,v} \in S$, and $a_{s_{t,v},t,v} \in A_s$ are the exit point, the split point, and the autoencoder, respectively, selected by the CAV v at the time interval t.

Similarly to [25], the set of layers executed on the CAV is labeled as $\mathcal{L}_{z,v} = \{l_1, l_2, ..., l^{\mathrm{encode}}_{a_s}\}$ and the set of layers executed on the server as $\mathcal{L}_{z,b} = \{l^{\mathrm{decode}}_{a_s}, l_2, ..., l_e\}$, where l denotes the layer of the CNN. The layers denoted as $l^{\mathrm{encode}}_{a_s}$ and $l^{\mathrm{decode}}_{a_s}$ are the encoder and decoder parts of the autoencoder, respectively. The encoder and decoder components of the autoencoder employ convolutional layers, consistent with the remainder of the network's architecture. Finally, the layer l_e is

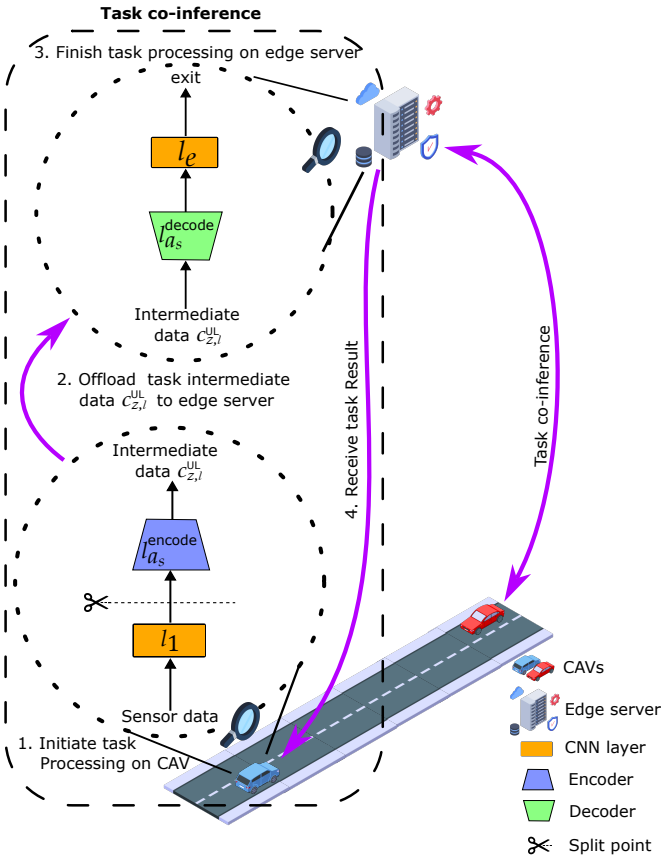


Fig. 1: The model shows CAVs communicating with an edge server. Each CAV sequentially processes CNN layers, ending with an autoencoder that compresses the final layer's output. The compressed data is offloaded to the edge server, where it is decompressed by the decoder to reconstruct the last layer's output before continuing the execution of the subsequent CNN layers.

TABLE I: Main Notations Used

Symbol	Definition
b	edge server
v	CAV $v \in \mathcal{V}$
z	task $z \in \mathcal{Z}_{t,v}$
$\mathcal{Z}_{t,v}$	$\mathcal{Z}_{t,v} = \{z \in \mathcal{Z}_{t,v} \mid z$ generated by CAV within time interval $t\}$
e	exit $e \in E$
s	s-th split $s \in S$ determining task partitioning
a	autoencoder $a \in A$
$\zeta_{t,v}$	strategy for e, s and a for CAV v withing time iterval t
$\mu_{t,v}$	accuracy of the CNN model for CAV v within time interval t
$\psi^{\text{lat}}_{t,v}$	task requirement for latency of task execution
$\psi^{\text{acc}}_{t,v}$	task requirement for accuracy of the CNN model
W^{lat}_v	weight for latency of task execution
W^{acc}_v	weight for accuracy of the CNN model
$r_{t,v}$	data rate between CAV v and edge server b
η_v	Computational capability of CAV v in FLOPS
η_b	Computational capability of edge server b in FLOPS
N^{RB}	Number of resource blocks available

the exit point, which corresponds to the end of the CNN model processing. The task co-inference of these layers, achieved through the interaction between the edge server and the CAV(s), is depicted in Fig. 1.

C. Communication Model

In this section, we introduce communication model employed if task is offloaded from the CAV to edge server. Following recent related works, such as [34], we assume the network conditions to be constant when the tasks are transferred to the edge server. This assumption is based on the fact that the task offloading process typically takes a very short amount of time (in order of a few milliseconds). Then, the data rate for offloading of the computation task from the CAV v to the edge server b at time t can be expressed as:

$$r_{t,v} = N^{\text{bit}}_v \rho_v \frac{N^{\text{RB}}}{|\mathcal{V}|} R, \tag{2}$$

where N^{bit}_v is the number of bits per symbol and ρ_v is the code rate for transmissions generated by the v-th vehicle, both parameters are determined by the modulation and coding scheme as referenced in [35]. The term N^{RB} denotes the total number of resource blocks available at the base station (i.e., the base station where the edge server is co-located) to the set of CAVs $|\mathcal{V}|$, and R represents the symbol rate of the base station.

In order to determine data volume to be offloaded for task $z \in \mathcal{Z}_{t,v}$ depending on the selected split and autoencoder in CNN described in previous subsection, we calculate the amount of bits of the layer l output as [36]:

$$c^{\text{UP}}_{z,l} = D^{\text{width}}_l D^{\text{height}}_l l^{\text{ch}} P, \tag{3}$$

where, D^{width}_l and D^{height}_l denote the spatial dimensions of the output of the layer l. In the context of a convolutional layer [37], these terms refer to the horizontal and vertical dimensions of the feature maps [37]. Conversely, for a linear layer [37], these values would typically be 1, reflecting the absence of a spatial structure in such layers. Furthermore, l^{ch} corresponds to the number of output channels, and P is the bit size according to the number format in float32 [38]. The value of l^{ch} is constant for each autoencoder and is preset before the training.

The D^{width}_l and D^{height}_l are recursively computed using [39]:

$$D_l = \begin{cases} \left\lfloor \frac{D_{l-1}+2(l^{\text{pad}})-(l^{\text{ker}})}{(l^{\text{str}})} - 1 \right\rfloor, & \text{if } l > 0, \\ D_0, & \text{otherwise,} \end{cases} \tag{4}$$

where D_l is either width D^{width}_l or height D^{height}_l of the l-th layer output, l^{pad} is the padding of the layer, l^{ker} is the kernel size of the layer, l^{str} is the stride of the layer, and D_0 is the dimension of the input image generated by the sensors on the CAV.

Finally, the uplink communication latency for the offloading of the task $z \in \mathcal{Z}_{t,v}$ is defined as:

$$t^{\text{comm}}_z = t^{\text{comm}}_{z,w} + \frac{c^{\text{UL}}_{z,l}}{r_{t,v}}, \tag{5}$$

where $t^{\text{comm}}_{z,w}$ denotes the latency that accounts for the queue and network dynamics (e.g., packet loss), as in [40]. We assume that the CAV generates new tasks and enqueues them in a First-In-First-Out (FIFO) order, as in other works, such as [41]. This assumption increases its generality, making it suitable for various wireless network scenarios [41]. Since the downlink data size is a single scalar value returned by processing of the CNN, it is negligible and communication for delivery of the computation results from the edge server to the CAV is neglected as in other related works, see e.g., [42]. We assume a flat channel during offloading process, similarly to [42], as small-scale fading effects are averaged out across the transmission latency. This modeling choice is reasonable for analyzing the fundamental trends of the problem under study, as flat channel models have been widely adopted in vehicular offloading studies (see e.g., [10], [43], [44], [45]).

D. Computation Model

Now, let's discuss computation model employed by our work. The overall computational demand in floating point operations (FLOP) for a part of the task z executed on the CAV v is expressed as:

$$I_{z,v} = \sum_{l \in \mathcal{L}_{z,v}} I_{z,l}. \tag{6}$$

where $I_{z,l}$ is computing demand of the CNN layer l processed at the CAV estimated in line with [46] as:

$$I_{z,l} = 2 D^{\text{width}}_l D^{\text{height}}_l (l^{\text{ker}})^2 l^{\text{ch}} (l-1)^{\text{ch}}, \tag{7}$$

where $(l-1)^{\text{ch}}$ is the number of input channels of the layer l.

Subsequently, the latency of the task execution on CAV is determined as:

$$t^{\text{comp}}_{z,v} = t^{\text{comp}}_{z,w,v} + \frac{I_{z,v}}{\eta_v}, \tag{8}$$

where $t^{\text{comp}}_{z,w,v}$ denotes the waiting (queuing) time for the completion of previous tasks on the CAV, and η_v indicates the computation capacity of the CAV, measured in floating point

operations per second (FLOPS). Note that similarly as with communication resources, we exploit FIFO for queuing the tasks on the CAV.

The task can be offloaded to the edge server for further processing. The computational demand of a part of the task z processed on the edge server b is expressed as:

$$I_{z,b} = \sum_{l \in \mathcal{L}_{z,b}} I_{z,l}, \tag{9}$$

where computational demand of a single layer l are estimated according to (7).

The overall computation latency on the edge server $t^{\text{comp}}_{z,b}$ is derived as:

$$t^{\text{comp}}_{z,b} = t^{\text{comp}}_{z,w,b} + \frac{I_{z,b}}{\eta_b}, \tag{10}$$

where $t^{\text{comp}}_{z,w,b}$ signifies the cumulative latency incurred by all preceding tasks on the edge server, and η_b represents the computing capacity of the edge server b.

Finally, we can calculate the total computation latency of the task execution consisting of the local execution time on the CAV and the remote execution time on the edge server as:

$$t^{\text{comp}}_{z} = t^{\text{comp}}_{z,v} + t^{\text{comp}}_{z,b}. \tag{11}$$

E. CAV Utility

The utility of the CAV refers to the measure of how well the CAV chooses the optimal strategy under different environmental states (e.g., data rate between the CAV and edge server) [47], [48]. The optimal strategy is, then, updated at the beginning of each time interval t. We model utility as a trade-off between the latency and the accuracy that the CAV achieves relative to the task's requirements.

To define the latency, we first specify the total latency of the task t_z, which comprises of the communication latency t^{comm}_z and the computation latency t^{comp}_z, as follows:

$$t_z = t^{\text{comm}}_z + t^{\text{comp}}_z. \tag{12}$$

Since an unknown number of tasks can be executed during each time interval t, we quantify the latency as an average latency $t_{t,v}$ over all tasks generated in t as:

$$t_{t,v} = \frac{\sum_{z \in \mathcal{Z}_{t,v}} t_z}{|\mathcal{Z}_{t,v}|}, \tag{13}$$

Similarly, we define the accuracy as a ratio between the number of correctly classified tasks depending on the current strategy within certain time period t ($\kappa_{t,v}$) to all generate tasks as:

$$\mu_{t,v} = \frac{\kappa_{t,v}}{|\mathcal{Z}_{t,v}|}, \tag{14}$$

Note that in the event the task is discarded, such task is deemed to have incorrectly classified image.

Now, we define a reward function for the time interval t and the vehicle v as:

$$u_{t,v} = (1 - \frac{t_{t,v}}{\psi^{\text{lat}}_{t,v}}) W^{\text{lat}}_v + \frac{\mu_{t,v} - \psi^{\text{acc}}_{t,v}}{1 - \psi^{\text{acc}}_{t,v}} W^{\text{acc}}_v, \tag{15}$$

where $\psi^{\text{lat}}_{t,v}$ and $\psi^{\text{acc}}_{t,v}$ are the requirements for the latency and the accuracy, respectively, and W^{lat}_v and W^{acc}_v are the weights for the latency and the accuracy, respectively. The weights W^{lat}_v and W^{acc}_v are crucial in defining the trade-off between latency and accuracy, and are determined by the application's requirements. The application's priorities dictate whether the latency or the accuracy is more important. The first part of (15) quantifies the reward's latency value, whereas the second part delineates the reward's accuracy value. Both parts of (15) have two properties: *i)* they are normalized to the same range (e.g., between [0, 1]), and *ii)* they are positively correlated with the degree of satisfaction of the task's requirements. In other words, the first part of (15) increases with lower latency, while the second part of (15) increases with higher accuracy, relative to the task requirements.

The reward function defined in (15) assigns a positive or negative value to the latency and the accuracy depending on whether it meets or violates requirements for these values. The extent of deviation from the requirements directly corresponds to the magnitude of these performance metrics. Note, that the reward function can achieve a positive value even if one of the requirements is not met, depending on the weights.

Now, knowing the reward function, we also define a utility for a CAV v as a sum of rewards over an infinite time period as:

$$\Phi_{t,v} = \sum_{t \in [0,\infty]} \gamma_{t-1} u_{t,v}, \tag{16}$$

where $\gamma_{t-1} \in [0, 1]$ is a discount factor, that determines the present value of future utilities [49]. The discount factor is one of the hyperparameters that is typically tuned using numerical methods, such as hyperparameter search [50]. The utility of our system depends on how well it achieves the highest possible accuracy and the lowest possible latency over a infinite period. We consider an infinite period in our problem due to its classification as an infinite-horizon problem [51]. In such scenarios, the decision-making process extends indefinitely into the future. However, this utility is not solely dependent on the accuracy and latency of a single CAV, but it is influenced by all CAVs in the system. When the CAV selects the strategy $\zeta_{t,v}$ defined in (1), it invariably impacts all other CAVs in the system. Conversely, the utility of the CAV is also influenced by the strategies of all other CAVs, denoted as $\zeta_{t,\mathcal{V}'}$, where $\mathcal{V}' = \mathcal{V} - \{v\}$.

III. Problem Formulation

Our objective is to maximize the utility of all CAVs. We formulate such objective as:

$$\text{P1:} \quad \max_{\{\zeta_{t,v}, \zeta_{t,\mathcal{V}'}\}} \sum_{v \in \mathcal{V}} \Phi_{t,v} \tag{17}$$

$$\text{s.t.} \quad s_{t,v} \in S,\ a_{s_{t,v},t,v} \in A_s,\ e_{t,v} \in E, \tag{17a}$$

$$I_{z,v}, + I_{z,b} = I_z, \quad \forall z \in \mathcal{Z}_{t,v} \tag{17b}$$

$$\mu_{t,v} \geq \psi^{\text{acc}}_{t,v}, \quad \forall v \in \mathcal{V} \tag{17c}$$

$$t_{t,v} \leq \psi^{\text{lat}}_{t,v}, \quad \forall v \in \mathcal{V} \tag{17d}$$

where constraint (17a) ensures that the split, autoencoder, and exit are confined within predefined sets; constraint (17b) guarantees that the sum of instructions executed on the CAV and on the edge server is equivalent to all instructions that

need to be executed; constraint (17c) guarantees that for all CAVs, accuracy remains within the bounds defined by the task's accuracy requirements; and constraint (17d) ensures that average latencies for all CAVs adheres to the task's specified latency requirement.

The optimization problem formulated in (17) is a mixed-integer nonlinear programming (MINLP) problem. Such problems are known to be NP-hard, thus very difficult to be solved in polynomial time by conventional optimization techniques, such as simplex or interior-point methods. Consequently, the MINLP problems generally require the iterative numerical methods to find approximate solutions. Moreover, the problem becomes even more complicated if the number of CAVs in the environment increases. The strategies of the agents affect the communication and computational resources available, which introduces uncertainty and complexity to the problem. Furthermore, the agents' strategies are interdependent, which means that the optimal solution for one agent depends on the actions of the others. Our objective is to find the strategy that satisfies the utility function defined in (17). However, this is a difficult multi-agent optimization problem that involves non-linear and non-convex constraints. Hence, to address this problem, we apply a deep RL algorithm called Multi-Agent Deep Deterministic Policy Gradient (MADDPG), which is a state-of-the-art method for solving complex multi-agent scenarios.

IV. Proposed Approach

In this section, we first propose several modifications to the CNN architectures (e.g., VGG-16 architecture based on [52]) taking place during the offline configuration of the CNN in order to jointly incorporate early exits, splits, and autoencoders. These components allow us to adjust the inference process according to the environmental conditions, such as the data rate to the base station. Then, we describe MADDPG and its adaptation to our problem. After that, we transform the problem to Markov decision process and device a MADDPG-based strategy selection algorithm. Last, we also discuss several practical implementation aspects of proposed approach.

A. Offline Configuration of CNN

In this section, we propose a necessary enhancements facilitated during an offline configuration of CNN in order to provide a dynamic execution of CNN while enabling a co-inference paradigm between the CAVs and the edge server, illustrated in Fig. 1. In particular, our goal is to integrate early exits, splits and autoencoders within our CNN network to enable their joint selection. The incorporation of early exits and splits allows us to determine the number of layers to be processed prior to the termination of CNN execution and potential offloading of intermediate data to the edge server, respectively. On top of that, the selection of potential split point is strongly impacted by choosing a degree of data compression employed during the offloading process thanks to the set of trained autoencoders. All these features allows us to find a proper trade-off between the CNN's computational complexity and it's accuracy. The offline configuration, featuring the modified CNN model, is depicted in Fig. 2.

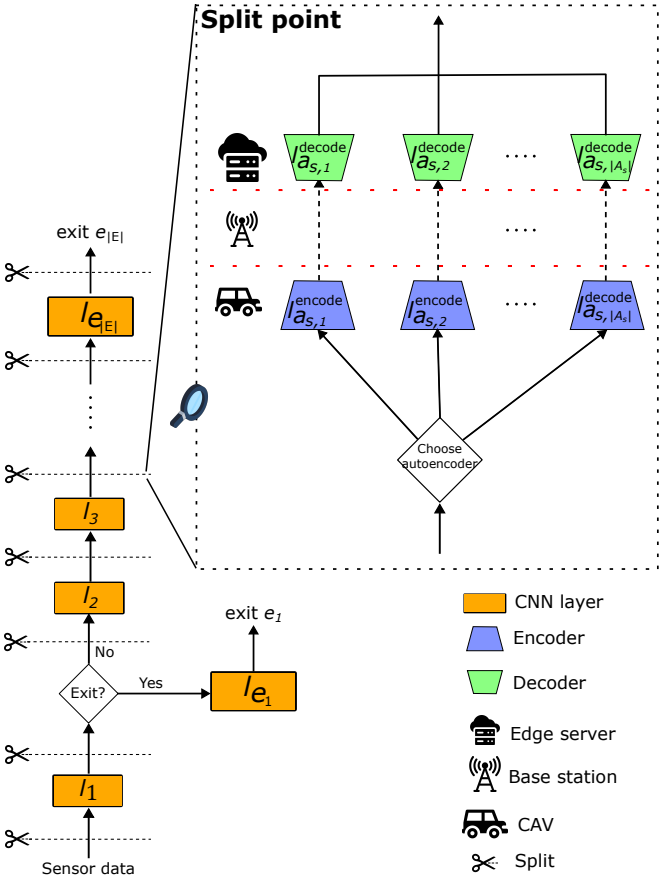


Fig. 2: Schematic representation of a CNN architecture, which is augmented with additional splits and early exit mechanisms. Within the proposed model, each split is equipped with a selection of autoencoders from which a single autoencoder is chosen based on current strategy. The encoder segment of this autoencoder is then tasked with encoding the last executed layer of the network. Once the data is encoded, it is transferred to an edge server. Subsequently, the decoder segment of the same autoencoder decodes the data, reconstructing the state of the last executed layer.

To augment the CNN with additional branches and, thus, having a possibility of early execution termination, we employ the BranchyNet principle [27]. To this end, we train jointly the auxiliary exits together with the main sequence of layers (i.e., excluding those belonging to the early exit branches), commonly referred to as backbone network. Furthermore, utilizing a technique known as early stopping [53] allows us to refine the training process. This strategic approach permits CNN to terminate processing at intermediate stages, particularly when subsequent computations would excessively increase latency.

Moreover, the establishment of layer splits, as proposed in [19], after CNN layers, facilitates the offloading of intermediate data to the edge server, where further computation follows. This proves particularly beneficial in scenarios where CAVs produce a substantial volume of tasks. The layer split architecture empowers us to harness the computational capabilities inherent to the CAVs, allowing for the partial processing of tasks on-board. This, in turn, enables the edge server to accommodate an

increased number of CAVs, each generating an extensive array of tasks.

The efficiency of split computing is constrained by high volume of the data generated by each intermediate layer [19], rendering many splits impractical. To address this problem, we enhance the CNN by autoencoders to encode the intermediate data prior to offloading, thereby enhancing the usability of the splits. Nonetheless, the compression facilitated by autoencoders is inherently lossy, which invariably impacts the accuracy of our CNN. In our novel approach, we deploy a set of autoencoders, trained to reconstruct data, each calibrated to provide varying degrees of compression and, correspondingly, different levels of accuracy loss. More specifically, we introduce a dedicated set of autoencoders for each split point, thereby enhancing the execution flexibility of our CNN architecture and increasing the number of viable split points for selection.

To jointly select the most suitable exit point, split, and autoencoder for each CAV according to current environmental conditions, we employ the MADDPG process, introduced in the following subsections.

B. Preliminaries MADDPG

The proposed innovative cooperative framework, incorporating the enhanced CNN (see previous section), employs the MADDPG process to enable joint cooperative selection of early exits, splits and autoencoders. Hence, in this section, we describe first the MADDPG and its adaption to our problem. In fact, the MADDPG is an extension of Deep Deterministic Policy Gradient (DDPG), which is an actor-critic method learning from past experience while exploiting different policies for exploration and exploitation. More specifically, the DDPG is a model-free, off-policy algorithm that uses soft updates and the Bellman equation for training in order to maximize its own reward in single-agent settings. The MADDPG, then, extends DDPG to multiple agents system and, based on local observations for action selection and global observations for value estimation, enables a decentralized execution and centralized training.

In a multi-agent environment, each agent is represented as the CAV with a policy network and a target policy network, denoted as $\pi_v(o_{t,v}|\theta_{\pi_v})$ and $\pi'_v(o_{t,v}|\theta_{\pi'_v})$, respectively, where $o_{t,v}$ is the local observation of the CAV, θ_{π_v}, and $\theta_{\pi'_v}$ are the network parameters that define the current and the target policy, respectively. Each agent also has a critic and a target critic, which represent the centralized action-value function. These critics are denoted as $Q_v(o_t, \zeta_{t,1}, ..., \zeta_{t,|\mathcal{V}|}|\theta_{Q_v})$ for the critic network and $Q'_v(o_t, \zeta_{t,1}, ..., \zeta_{t,|\mathcal{V}|}|\theta_{Q'_v})$ for the target critic network, where o_t represents the state for the critic. In proposed approach, the o_t represents the observations of all the CAVs as $o_t = (o_{t,1}, o_{t,2}, ..., o_{t,|\mathcal{V}|})$. To train the policy network π of the actor, we sample a replay buffer of S samples as $(o_j, \zeta_j, u_j, o'_j)$, where o'_j is the next state, and optimize the sampled gradient of the expected return for the agent v (e.g., $J(\theta_{\pi_v}) = \mathbb{E}[\Phi_v]$) as:

$$\nabla_{\theta_{\pi_v}} J \approx \frac{1}{S}\sum_j \nabla_{\theta_{\pi_v}} \pi_v(o_{j,v}) \nabla_{\zeta_v} Q_v(o_j, \zeta_{j,1}, ..., \zeta_{j,v}..., \zeta_{j,|\mathcal{V}|}) \mid_{\zeta_{j,v}=\pi_v(o_{j,v})} . \quad (18)$$

We optimize the centralized action-value function network of the critic by minimizing the loss function defined as [54]:

$$\mathcal{L}(\theta_{Q_v}) = \frac{1}{S}\sum_j (y_j - Q_v(x_j, \zeta_{j,1}, ..., \zeta_{j,|\mathcal{V}|}))^2, \quad (19)$$

where

$$y_j = u_{j,v} + \gamma Q'_v(o'_j, \zeta'_{j,1}, ..., \zeta'_{j,|\mathcal{V}|}) \mid_{\zeta'_{j,v}=\pi'_v(o_{j,v})} . \quad (20)$$

After training the actor and the critic, we softly update our target networks for the actor and the critic as follows:

$$\begin{aligned} \theta_{\pi'_v} &\leftarrow \tau\theta_{\pi_v} + (1-\tau)\theta_{\pi'_v}, \\ \theta_{Q'_v} &\leftarrow \tau\theta_{Q_v} + (1-\tau)\theta_{Q'_v}, \end{aligned} \quad (21)$$

where τ is a soft update coefficient, typically set close to 0.

C. Formulating the Problem as Markov Decision Process

Before delving deeply into the description of the proposed MADDPG-based strategy selection algorithm, we transform the problem into a Markov Decision Process (MDP). This transformation is crucial as MDPs facilitate the optimization of sequential decisions (i.e., strategy selection) by enabling the learning of policies that maximize the utility function, as defined in (17). The MDP is characterized by elements such as agents, observations, actions, and rewards. These components are delineated in a subsequent list, detailing their specific definitions and functions within the MDP framework.

1) Agents: In the context of the MDP framework adopted in this paper, the agents are represented by CAVs. The complete set of CAVs is denoted by $\mathcal{V}$, and each CAV $v \in \mathcal{V}$ is considered an individual decision-making agent within the MDP.

2) Observation Space: Upon the completion of the time interval t, we compute the average latency of task execution $t_{t,v}$ as defined in (13), the average time of task offloading $t^{\text{comm}}_{t,v}$, the data rate in the current time interval $r_{t,v}$, and the task-specific requirements for latency $\psi^{\text{lat}}_{t,v}$ and accuracy $\psi^{\text{acc}}_{t,v}$. The observation for CAV for the time interval t is subsequently defined as:

$$o_{t,v} = \{t_{t,v}, t^{\text{comm}}_{t,v}, r_{t,v}, \psi^{\text{lat}}_{t,v}, \psi^{\text{acc}}_{t,v}\}. \quad (22)$$

The observation space for a CAV is the set of all such possible observations it can encounter. In the MADDPG framework, while each agent's actor network selects actions based on its individual observations, the critic network has access to the joint observation space. This joint space, which is the set of all possible combinations of observations from all agents, is utilized during the training phase to evaluate the joint action values and update the actor networks. The joint observation for the time interval t, denoted as o_t, is the aggregation of the individual observations of all CAVs, constructed as follows:

$$o_t = \{o_{t,1}, o_{t,2}, ..., o_{t,|\mathcal{V}|}\}. \quad (23)$$

It is important to note that while the critic uses the joint observation space for training, each agent's actor network makes decisions based solely on its own observations during execution.

3) Action space: At the beginning of the time interval t, each CAV within the set $\mathcal{V}$ selects its strategy as delineated in (1), which signifies the actions of the agents. Action space for each agent represents the all possible values agent can choose from, when selecting an action. During the training phase, it is imperative to consider the actions of all other CAVs. Consequently, the joint action of all CAVs at time interval t is defined as:

$$\zeta_t = \{\zeta_{t,1}, \zeta_{t,2}, ..., \zeta_{t,|\mathcal{V}|}\}. \tag{24}$$

This represents a specific instance of a joint action. In contrast, the joint action space refers to the set of all possible joint actions that the collective of CAVs could take.

4) Reward: To optimize our utility function, we employ a reward function to evaluate the performance efficacy of the agent, as defined in (15).

D. MADDPG-based Strategy Selection

Algorithm 1 MADDPG-based strategy selection algorithm.

Require: $|\mathcal{V}| > 1$

1: $T_{max} \leftarrow$ maximum number of steps
2: Initialize actor networks $\pi_v(o_v|\theta_\pi)$ and critic networks $Q_v(o, \zeta|\theta_Q)$ for each agent $v \in \mathcal{V}$ with random weights θ_π and θ_Q
3: Initialize target networks $\pi'_v(o_v|\theta_{\pi'})$ and $Q'_v(o, \zeta|\theta_{Q'})$ for each agent $v \in \mathcal{V}$ with weights $\theta_{\pi'} \leftarrow \theta_\pi$ and $\theta_{Q'} \leftarrow \theta_Q$
4: Initialize replay buffer $\mathcal{D}$
5: observe initial state o_0
6: $\mathcal{P} \leftarrow$ init Ornstein-Uhlenbeck random process
7: **while** $t \leq T_{max}$ **do**
8: **for** $v \in \mathcal{V}$ **do**
9: Observe local observation $o_{t,v}$ from state o_t
10: Take action $\zeta_{t,v} = \pi_v(o_{t,v}|\theta_{\pi_v}) + \mathcal{P}$ according to actor network and noise
11: **end for**
12: Execute actions of all agents: $\zeta_t = (\zeta_{t,1}, ..., \zeta_{t,|\mathcal{V}|})$
13: observe reward u^t and next state o'_t
14: Store $(o_t, \zeta_t, u_t, o'_t)$ into the replay buffer $\mathcal{D}$
15: **for** $v \in \mathcal{V}$ **do**
16: Sample a random minibatch of S samples $(o_j, \zeta_j, u_j, o'_j)$ from replay buffer $\mathcal{D}$
17: Update critic by minimizing the loss (19).
18: Update actor using the sampled policy gradient (18).
19: **end for**
20: **for** $v \in \mathcal{V}$ **do**
21: Softly update parameters of target networks (21).
22: **end for**
23: $t \leftarrow t + 1$
24: **end while**

The MDP is modeled as a discrete-time stochastic control process, and a training algorithm based on MADDPG is proposed to find the optimal policy π^*_v representing a policy towards which the CAVs are trying to optimize iteratively over the course of T_{max} time steps.

The MADDPG-based strategy selection algorithm is detailed in Algorithm 1. In the initial phase, the number of time steps T_{max} is defined (line 1), which delineates the training duration for the agents. The actor and critic networks, along with their target networks, are established to facilitate the subsequent optimization process (lines 2 and 3). A replay buffer is also created to store tuples of state, action, reward, and next state (line 4). Before the start of the training process we observe initial state o_0 of the environment and initiate the Ornstein-Uhlenbeck random process [55] (lines 5 and 6). This noise process is chosen for its ability to introduce a controlled amount of randomness into the agent's actions, which is essential for exploring the action space effectively. The temporally correlated nature of the Ornstein-Uhlenbeck process ensures that exploration is conducted in a manner that allows the agent to discover and learn from a diverse range of experiences without making drastic, random jumps that could lead to suboptimal learning.

Then, at each time step t and for each CAV, the observations of all agents are recorded as the current state (line 9). To facilitate exploration, an Ornstein-Uhlenbeck noise process $\mathcal{P}$ is applied to the action. The actions $\zeta_{t,v}$ are obtained from the actor network with parameters θ_π for each agent, based on individual observations (line 10). The action is represented by a set of categorical values indicating the index of split, autoencoder, and exit. To discretize the continuous output of the actor network, multiple outputs are utilized for each category. Following the approach in [56], Gumbel Softmax is applied to three groups of outputs to select the appropriate category. The noise $\mathcal{P}$ is added to the action, and as the agent accumulates experience and improves its policy, the exploration rate is gradually decreased. This reduction is crucial for transitioning the agent's strategy from a phase of broad exploration to a more targeted exploitation of the learned policy, thereby enhancing the efficiency and effectiveness of the decision-making process. Following the actions' execution (line 12), the next state o'_t and the reward u_t are observed (line 13). The transition data, which encapsulates state o_t, action ζ_t, reward u_t and next state o'_t, is stored in the replay buffer (line 14). The process is repeated until there is sufficient data in the replay buffer for training. Subsequently, a batch of data is sampled from the replay buffer, and the critic and actor networks are updated using (18) and (19) (lines 15-19). The target networks are softly updated using (21) (line 21).

In proposed MADDPG-based approach, we perform training updates at regular intervals rather than after every time step. Specifically, the training occurs after a predetermined number of environment interactions. During these intervals, the agents collect experience by interacting with the environment, and then the neural networks are updated using this accumulated experience.

E. Implementation Aspects of the Proposal

In this part we briefly discuss the required modifications to CNN, training process of MADDPG, and complexity.

1) Required modifications to CNN: The modifications of the CNN follow well-established patterns in deep learning, where each split point uses standard autoencoder pairs (e.g., [23]) and early exits implement conventional classification heads (e.g.,

[27]). The approach is modular, reusing similar architectural patterns across different points, making the implementation straightforward. Furthermore, these modifications to the CNN models are done during training of the CNN models and, therefore, before CNN model's deployment, allowing us to allocate more time to the modification of these neural network (i.e., inserting and training early exits and autoencoders).

2) Data Synchronization: The training process of the proposed MADDPG-based strategy can be run directly at each CAV. This approach would, however, cause substantial communication overhead due to the necessity of synchronizing replay buffers across all the CAVs. Consequently, we propose that the training of MADDPG-based algorithm is conducted rather on the edge server. In such a case, CAVs only need to transmit selected observational data (i.e., the average task latency $t_{t,v}$), from the complete set of observations $o_{t,v}$ defined in (22).

Additional data for observation $o_{t,v}$, can be observed directly from the edge server (i.e., average offloading latency $t^{\text{comm}}_{t,v}$ and data rate $r_{t,v}$). Observation $o_{t,v}$ also contains latency requirement $\psi^{\text{lat}}_{t,v}$ and accuracy requirement $\psi^{\text{acc}}_{t,v}$, however, these are only small numerical values. Furthermore, these requirements are modified solely as necessary, allowing for the transmission of these requirements to the edge server only when they change their value. As a result, obtaining the observation $o_{t,v}$ after each time step requires the transmission of only a few tens of bytes.

Upon receiving a new observation $o_{t,v}$ from the CAVs, their respective actors π_v predict new strategies $\zeta_{t,v} = \pi_v(o_{t,v}|\theta_{\pi_v})$. Therefore, we need to transmit the new strategies $\zeta_{t,v}$ to the CAVs. Given that these are three small integer values, they also amount to only a few tens of bytes per CAV. Hence, the data exchange required for strategy $\zeta_{t,v}$ updates is minimal and can be disregarded when compared to the larger volume of data involved in CNN task offloading.

3) Complexity Analysis: The time complexity of our algorithm is predominantly influenced by the execution speed of the neural networks. Analogous to the methodology presented in [57], we compute the time complexity of neural networks as a matrix multiplication operation between each layer. Given that we are working with linear layers in our actor and critic networks, we are essentially performing multiplication operations between two one-dimensional matrices (or vectors). Consequently, the time complexity of transitioning from layer i to layer j is $O(ij)$. Considering that we are operating with minibatches comprising S samples, it is imperative to account for the fact that each multiplication operation will be executed S times. Therefore, the complexity for layers i and j would be $O(ijS)$.

To traverse both networks consisting of I and J layers for the actor and critic networks respectively, we can express the complexity of training both as:

$$O((\sum_{i=0}^{I-1} u_{a,i}u_{a,i+1} + \sum_{j=0}^{J-1} u_{c,j}u_{c,j+1}) * S), \tag{25}$$

where $u_{a,i}$ and $u_{c,j}$ represent the number of neurons in the actor and critic networks of layers i and j, respectively. In a production environment, this complexity is reduced as only a single forward propagation of an actor is required, which can be represented as:

$$O(\sum_{i=0}^{I-1} u_{a,i}u_{a,i+1}). \tag{26}$$

To summarize, the proposal can easily be used in real-world scenarios since the duration of the decision-making process (usually in the order of tens of nanoseconds) is negligible when compared to the total latency of task execution (usually in the order of milliseconds).

V. Simulation Results

In this section, we evaluate performance of the proposed approach via simulations. First, we detail the simulation setup in Section V-A. Competitive state-of-the-art works considered for evaluation are discussed in Section V-B. Next, we present a comparative analysis of various CNN models (i.e., models representing our computational task) in Section V-C. Subsequently, we conduct an ablation study to assess the impact of omitting specific components of our algorithm in Section V-D. Simulation results for different levels of workload intensity, latency requirement, and available resource blocks in Sections V-E, V-F, and V-G, respectively.

A. Modeling and setting for evaluation

Inspired by other works, such as [25], [26], we utilize the VGG-16 architecture [52] and the AlexNet architecture [58], as a model of the computer vision task based on CNN for offloading. We select these models due to their well-established nature and extensive study in the field, see e.g., [59], [60], demonstrating that these architectures remain relevant in related studies. However, the proposed approach can be extended to any computer vision task and deep learning model, including transformer-based models that can be adapted for splitting and early exiting capabilities, as comprehensively discussed in [30]. The VGG-16 structure comprises 16 layers, including 13 convolutional and 3 linear layers. Furthermore, CNN incorporates 5 average pooling layers. The CNN is trained on the CIFAR-10 dataset [61], a common benchmark in the image classification. The AlexNet architecture consists of 8 layers, featuring 5 convolutional layers, and 3 fully connected layers. AlexaNet incorporates 3 max pooling layers and uses dropout for regularization.

In the offline configuration, we enhance the CNN models with three early exits approximately at 25%, 50%, and 75% of the model's computational complexity (i.e., after 2nd, 4th, and 7th convolutional layer) for VGG-16 architecture. Given AlexNet's smaller size, we implement a single early exit at 50% of computational complexity, specifically after the 3rd layer. Additionally, we introduce 21 splits across the VGG-16 CNN model and 10 splits across the AlexNet CNN model, ensuring a split after each convolutional layer, first pooling layers, input and output layers (note, that the output layer implies that the entire computational task is processed locally by the CAV) and after each early exit, as shown in Fig. 2. The decision to exclude the splits following the final average pooling layer and in-between linear layers is done due to their relatively lower computational demand when compared with the convolutional layers. Furthermore, both CNN models include eight autoencoders at each split,

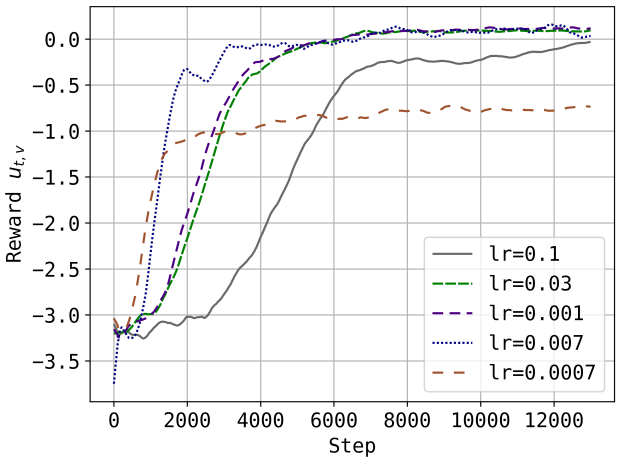


Fig. 3: Rewards for proposed method during training process under different learning rates values, where actor and critic had the same learning rate and with $|\mathcal{Z}_{t,v}| = 150$, $W_v^{\text{acc}} = 0.9$, $W_v^{\text{lat}} = 1$, $\psi_{t,v}^{\text{lat}} = 6.5$ms.

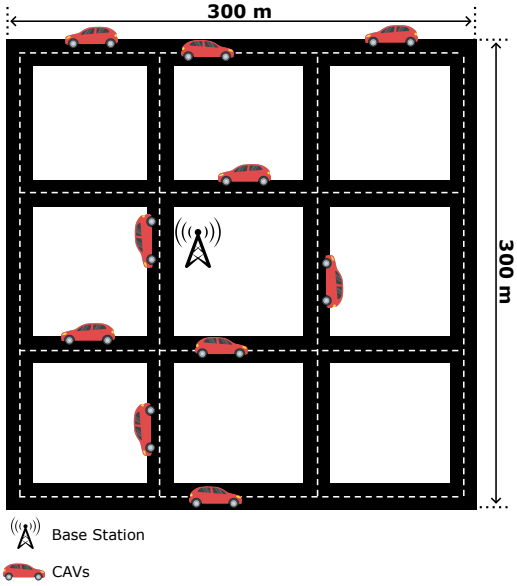


Fig. 4: Schematic representation of the analyzed urban traffic scenario for result evaluation.

each autoencoder with a unique compression level. Note, that we select eight autoencoders after extensive testing of various autoencoders, utilizing similar technique as [23]. Selection of eight autoencoders provides the most suitable trade-off between accuracy and latency. Additionally, this quantity is a good choice for applying MADDPG to effectively find the optimal policy. A smaller number would not offer sufficient variability for robust policy development while a larger count would complicate the action space, hindering the strategy optimization.

Similarly to [62], we fine-tune the MADDPG algorithm by employing grid search methodology to identify optimal hyperparameters. The results of the search for the optimal learning rate are illustrated in Fig. 3. Note that we chose to illustrate the effect of the learning rate, as it is widely recognized as one of the most crucial and extensively studied hyperparameters in reinforcement learning. This significance is well-documented in relevant research literature (e.g., [63]). It is observed that the values lr = 0.001, lr = 0.0007, and lr = 0.003 yield the highest rewards. Initially, these parameters are assigned high negative values to ensure that the agents are penalized should they diverge significantly from the specific requirements of the task. Notably, all three learning rate values mentioned above attain positive rewards at around 6000 steps, indicating that commendable balance between accuracy and latency is established. The actor network's architecture is structured as follows: It receives an input $o_{t,v}$ with $|o_{t,v}|$ input neurons, followed by three fully connected hidden layers of sizes 400, 300, and 300 neurons respectively. The output layer produces an action sequence $\zeta_{t,v}$, with the number of output neurons corresponding to size $|\zeta_{t,v}|$. The critic network features a larger input dimension of $|o_t|+|\zeta_t|$, where $|o_t|$ encompasses the combined input size $|o_{t,v}|$ for all vehicles $v \in \mathcal{V}$, and $|\zeta_t|$ represents the all actions $|\zeta_{t,v}|$ across all vehicles $v \in \mathcal{V}$. All layers in both networks are implemented as linear layers. This design choice is motivated by computational efficiency considerations - using (7), we can estimate that linear layers require fewer computational operations compared to convolutional alternatives. As a result, our actor network's execution latency is several orders of magnitude lower than our computational tasks (i.e., VGG-16 and AlexNet models) making this approach applicable to the real-world scenarios. The selected hyperparameter values are documented in Table II. The search for the optimal remaining hyperparameters was conducted in a manner analogous to that for the learning rate illustrated in Fig. 3.

Our CAVs operate within a Manhattan road network, characterized by the grid-like structure depicted in Fig. 4. This urban topology has been extensively studied (e.g., [64], [65]) due to its accurate representation of typical urban street layouts, providing a realistic framework for evaluating CAV performance in metropolitan environments. Within this network, CAVs movements are modeled using the Manhattan mobility model [66], which aligns with the grid-based nature of the environment. Demonstrating the results within an urban scenario presents a realistic and challenging environment for VEC, showcasing the robustness and applicability of the proposed approach. To simulate path-loss, the Hata model is employed. This model is recognized as a conventional model for the estimation of path-loss in both rural and urban settings [67].

A series of 10 independent simulations is conducted with each simulation of a duration of 2,000 time steps. This approach facilitates a comprehensive exploration of the stochastic processes under study. The execution of these 2,000 steps follows after the training of agents over 13,000 steps. The duration of 13,000 steps is a sufficient training period to accommodate the complexities of the environment and the agent's learning process. The full details of the system parameters are in Table III.

Our proposed approach is first evaluated for two different CNN models, followed by an evaluation through ablation analysis, and finally by varying: (i) workload intensity represented by the number of tasks generated per second $|\mathcal{Z}_v^t|$ by individual CAVs; (ii) the latency requirement $\psi_{t,v}^{\text{lat}}$; and (iii) communication

TABLE II: MADDPG hyperparameters

Parameter	Value
Actor learning rate	10^{-3}
Critic learning rate	10^{-3}
Soft update coefficient (τ)	0.01
Discount factor (γ)	0.9
Memory size	10^5
Batch size	64
Actor Network Configuration	$\{\lvert o_{t,v}\rvert$, 400, 300, 300, $\lvert\zeta_{t,v}\rvert\}$
Critic Network Configuration	$\{\lvert o_t\rvert + \lvert\zeta_t\rvert$, 400, 300, 300, 1$\}$

TABLE III: Simulation setup

Parameter	Value
Number of resource blocks (N_{RB})	100, 200
Symbol rate (R)	$2 * 10^9$
Computational power of CAV v (η_v)	$28.6 * 10^9$
Computational power of edge server b (η_b)	$41.3 * 10^{12}$
Number of tasks per t ($\lvert\mathcal{Z}_{t,v}\rvert$)	50, 180
Number of CAVs ($\lvert\mathcal{V}\rvert$)	10
Latency weight (W_v^{lat})	1, 0.5
Accuracy weight (W_v^{acc})	0.9, 0.25, 0.15
Accuracy requirement ($\psi_{t,v}^{\text{acc}}$)	0.82
Number of training steps	13,000
Number of simulation steps	2,000
Number of simulations	10

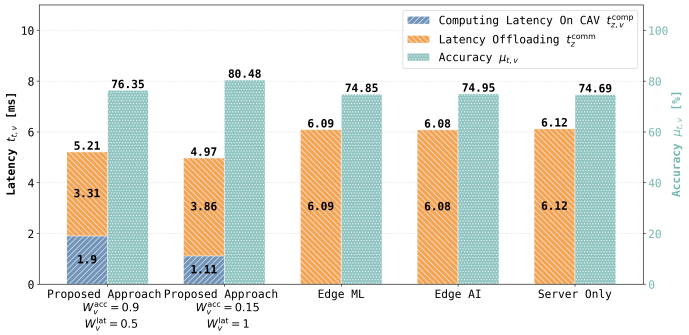
resource availability modeled via the available bandwidth of the connection (i.e., the number of available resource blocks N^{RB}). We also investigate the impact of the accuracy weight W_v^{acc} and latency weight W_v^{lat}, as these are crucial in determining the performance of the proposed algorithm and different values may offer advantages under various conditions.

B. Competitive state-of-the-art works

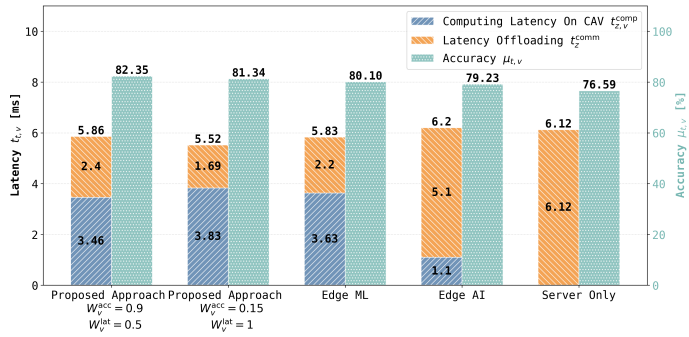
We compare the proposed approach with the following state-of-the-art benchmarks for the selection of split and early exit:

1) *Edge ML* [26]: *Edge ML* is a deep reinforcement learning-based approach that aims to minimize latency and energy consumption. It uses the DDPG algorithm, where the state space consists of latency, energy, and current transmission rate. The actions determine the confidence thresholds for early exits (i.e., confidence the prediction needs at early exits to stop execution), split point, and time interval for the selected strategy. Unlike our approach, *Edge ML* does not use autoencoders to create an artificial bottleneck for offloading data compression. Furthermore, the authors employ only a single-agent DDPG algorithm to determine the optimal strategy, hence, other agents are treated as non-stationary noise within the simulation. The *Edge ML* employs a DDPG-based algorithm, implying that the time complexity can be characterized in a manner analogous to our proposed approach.
2) *Edge AI* [25]: *Edge AI* is a search-based approach that iterates over all possible combinations of early exits and splits in the CNN. To estimate the execution time of the CNN on both the CAV and the edge server, the *Edge AI* baseline utilizes a regression model. This regression model is executed iteratively for each split point, first to determine the performance on the CAV and subsequently on the edge server. Searching from the last to the first exit and split points, *Edge AI* prioritizes the highest accuracy while meeting the latency requirements. Furthermore, no compression techniques, such as autoencoders, are employed by *Edge AI*. The *Edge AI* utilizes a conventional exhaustive search coupled with a regression model to predict the execution time of the model. In the worst-case scenario, the time complexity of this approach is $O((2d)l|E|)$, where d represents the number of features for the regression model (which is executed twice to predict the model execution time on both the edge server and the CAV), l denotes the number of layers, and $|E|$ signifies the number of exits of the CNN model. The time complexity of the *Edge AI* baseline exhibits polynomial growth, which is dependant upon the chosen CNN architecture (e.g., VGG16), representing our task. Specifically, an increase in the CNN architecture's size directly impacts the time complexity of *Edge AI* approach. In contrast, the proposed method demonstrates invariance to the size of the CNN architecture, maintaining consistent complexity regardless of the CNN architecture chosen.
3) *Server only*: In the *Server only* approach, input data is immediately offloaded to the server and processing of the whole taks is done at the server.

All the baselines and the proposed approach consider the same CNN architecture representing the computing task within the scope of simulations. This ensures comparability in the evaluation of the selected metrics related to the task offloading.



(a) Accuracy $\mu_{t,v}$ and latency $t_{t,v}$ evaluation on AlexNet architecture.



(b) Accuracy $\mu_{t,v}$ and latency $t_{t,v}$ evaluation on VGG-16 architecture.

Fig. 5: Cross-Model performance evaluation on VGG-16 and AlexNet architectures with $\psi_{t,v}^{\text{acc}} = 0.9$, $\psi_{t,v}^{\text{lat}} = 6.5$ms, $|\mathcal{Z}_{t,v}| = 150$ and $N^{\text{RB}} = 200$.

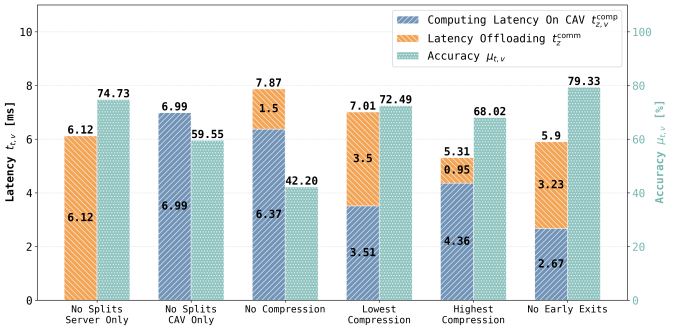


Fig. 6: Ablation Analysis on Proposed Approach with $W^{\text{lat}}v = 0.5$, $W^{\text{acc}}v = 0.9$, $\psi^{\text{acc}}_{t,v} = 0.9$, $\psi^{\text{lat}}_{t,v} = 6.5$ms, $|\mathcal{Z}_{t,v}| = 150$ and $N^{\text{RB}} = 200$.

C. Cross-Model Performance Evaluation

In this section, we compare AlexNet and VGG-16 architectures using requirements $\psi^{\text{acc}} * t, v = 0.9$ and $\psi^{\text{lat}} * t, v = 6.5$ ms. The proposed approach is compared against baselines using two weight configurations: accuracy-prioritizing ($W^{\text{lat}}v = 0.5$, $W^{\text{acc}}v = 0.9$) and latency-prioritizing ($W^{\text{lat}}v = 1$, $W^{\text{acc}}v = 0.15$).

Results are shown in Fig. 5. We evaluate accuracy $\mu_{t,v}$, CAV latency $t^{\text{comp}}z, v$, and offloading latency $t^{\text{comm}}z$. Server execution latency $t^{\text{comp}}z, b$ is omitted due to being several magnitudes lower than CAV processing latency $t^{\text{comp}}z, v$ and offloading latency t^{comm}_{z}.

The evaluation of AlexNet (Fig. 5a) shows the proposed approach achieves $t_{t,v} = 4.79$ ms latency and $\mu_{t,v} = 80.48\%$ accuracy, improving upon baselines by up to 18.79% and 7.76% respectively. While *Edge ML* and *Edge AI* determine that full offloading is more beneficial for AlexNet, the proposed approach uniquely leverages CAVs' computational capabilities through appropriate autoencoder selection and optimal split points.

When analyzing VGG-16 (Fig. 5b), the proposed approach achieves $t_{t,v} = 5.52$ ms latency and $\mu_{t,v} = 82.35\%$ accuracy, showing improvements up to 10.97% and 7.52% respectively. These results indicate that VGG-16 architecture, being larger and more complex model, presents a greater computational challenge that benefits more from split computing and early exiting. Unlike with AlexNet, even baseline approaches utilize CAVs' computational power for VGG-16, making it a more relevant benchmark for comparing the proposed approach against baselines. Consequently, for the remainder of the results, we consider the VGG-16 model as our computational task.

D. Ablation Analysis

In this section, we analyze the performance of our approach by systematically altering the capabilities of the VGG-16 CNN model. We perform an ablation study by selectively removing different components while keeping others intact. We begin by examining the system without splits, which consequently eliminates autoencoders but retains early exits. In this scenario, we consider two cases: always offloading to the server and computing the entire task on the device. Next, we restore splits but remove the capability of dynamically choosing autoencoders for compression. Instead, we present three fixed options: no compression, highest compression, and lowest compression. Last, we examine the system with splits and dynamic autoencoder selection, but without the capability of early exiting. This approach allows us to isolate the impact of each component on the overall system performance.

As shown in Fig. 6, the introduction of early exits without splits lowers accuracy compared to the *Server only* approach. This is because the system attempts to reduce latency, but since execution on the server is already fast, the latency reduction is minimal. When executing tasks on the CAV, we observe higher latency and lower accuracy due to the CAV's limited computational capabilities.

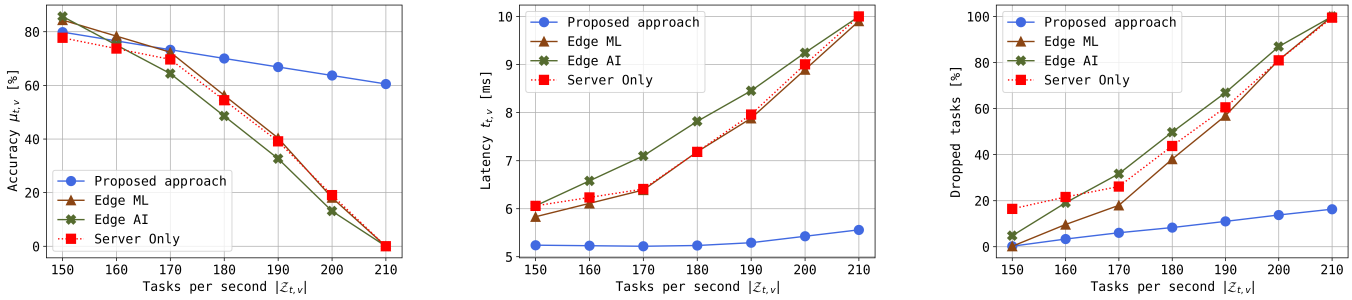
Our analysis of compression levels reveals that no compression yields the lowest latency and accuracy. This is since proposed approach executes most of the tasks on the CAV. Introducing compression significantly improves accuracy. The lowest compression setting still results in relatively high latency of 7.01 ms, while the highest compression setting trades off about 4.47% accuracy for a 1.7 ms latency improvement. These results suggest that excessive compression can significantly reduce the CNN model's (e.g., VGG-16) accuracy, while insufficient compression may result in unacceptable latency. Therefore, it is crucial to optimize the compression rate by carefully selecting an appropriate autoencoder, considering the application-specific latency and accuracy requirements. The proposed approach without early exits achieves performance similar to the that exhibited by the full proposed approach, as shown in Fig. 5b. These results demonstrate that the ability to select an appropriate early exit when necessary can significantly improve accuracy (by 3.02% in our tests) while also slightly reducing latency (by 0.04ms), highlighting the value of this feature within the proposed approach.

This analysis demonstrates that each component plays a crucial role in the performance of our proposed approach. The ablation study reveals that the combination of splits, dynamic autoencoder selection, and early exits significantly contributes to the system's overall performance and efficiency.

E. Workload intensity

At this stage, we systematically increase the number of tasks $|\mathcal{Z}_{t,v}|$ generated during each time interval t for all CAVs. Increasing the task generation rate necessitates a proportional reduction in latency requirements $\psi^{\text{lat}}_{t,v}$ to ensure the task processing remains uninterrupted. This adjustment is critical to prevent the task overflow and maintain operational integrity, particularly as the volume of tasks scales up. The latency requirement is thus calibrated to the task generation frequency (i.e., $\psi^{\text{lat}}_{t,v} = \frac{t}{|\mathcal{Z}_{t,v}|}$), enabling the system to accommodate a higher throughput of tasks without compromising performance. The outcomes of the simulations are depicted in Fig. 7.

As depicted in Fig. 7a, an increase in the number of generated tasks $|\mathcal{Z}_{t,v}|$ cause a decline in the accuracy of the task processing $\mu_{t,v}$. This fact is attributed to the algorithm's selection of autoencoders with elevated compression ratios to accommodate the additional tasks. Moreover, an increase in demands correlates with an increase in the volume of

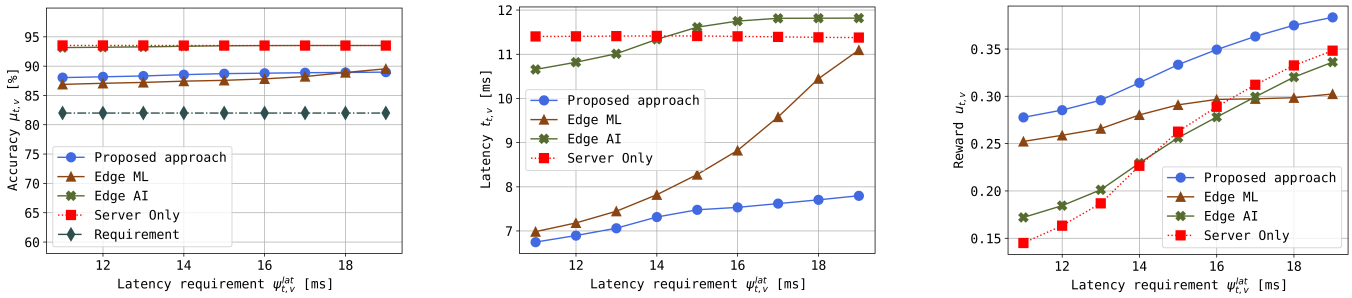


(a) Accuracy $\mu_{t,v}$ under different levels of tasks generated per second $|\mathcal{Z}_{t,v}|$.

(b) Latency $t_{t,v}$ under different levels of tasks generated per second $|\mathcal{Z}_{t,v}|$.

(c) Dropped tasks ratio under different levels of tasks generated per second $|\mathcal{Z}_{t,v}|$.

Fig. 7: Impact of workload intensity in tasks per second $|\mathcal{Z}_{t,v}|$ with $W_v^{\text{lat}} = 1$, $W_v^{\text{acc}} = 0.15$ and $N^{\text{RB}} = 200$.



(a) Accuracy $\mu_{t,v}$ under different levels of latency requirement $\psi_{t,v}^{\text{lat}}$.

(b) Latency $t_{t,v}$ under different levels of latency requirement $\psi_{t,v}^{\text{lat}}$.

(c) Reward under different levels of latency requirement $\psi_{t,v}^{\text{lat}}$.

Fig. 8: Performance evaluation at different levels of latency requirement $\psi_{t,v}^{\text{lat}}$ with $W_v^{\text{lat}} = 0.5$, $W_v^{\text{acc}} = 0.25$, $|\mathcal{Z}_{t,v}| = 50$ and $N^{\text{RB}} = 100$.

tasks that are dropped (refer to Fig. 7c), which consequently diminishes accuracy $\mu_{t,v}$. For instance, at the task generation rate $|\mathcal{Z}_{t,v}| = 210$, all baselines fail to maintain any level of accuracy. i.e., $\mu_{t,v} = 0\%$, whereas the proposed approach still achieves the accuracy of $\mu_{t,v} = 60.5\%$. This disparity arises from the inability of other methods to manage the increased demand, resulting in the dropout of all tasks. Conversely, when the task generation rate $|\mathcal{Z}_{t,v}| = 150$, the proposal slightly underperforms *Edge AI* and *Edge ML* in terms of accuracy. This is due a result of the prioritization of tasks latency minimization by the proposal.

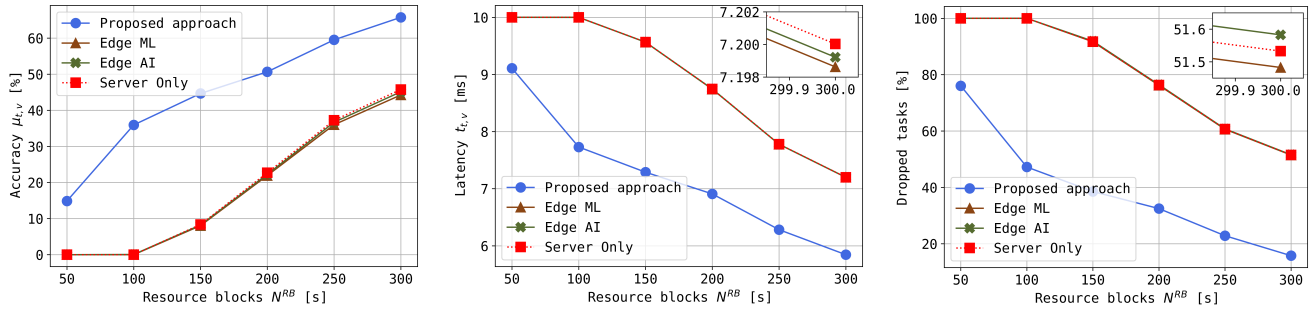
Furthermore, Fig. 7b demonstrates that the task latency $t_{t,v}$ for the proposal is reduced notably compared to all benchmarks. While the latency increases exponentially for all benchmarks, the proposal leads to relatively stable task latency $t_{t,v}$ with only a slow and almost linear increase with the number of generated tasks. The significant improvement in the latency by the proposal results the proposed multi-agent design, which allows CAVs to collaborate and compute larger segments of the task on CAV. The proposed method is able to achieve up to 44.4% lower latency $t_{t,v}$ compared to the baselines.

F. Latency Requirement

In the next set of simulations, we gradually reduce the latency requirements $\psi_{t,v}^{\text{lat}}$ to examine effects of the latency requirement $\psi_{t,v}^{\text{lat}}$ on the algorithms' performance (i.e., impact on accruacy $\mu_{t,v}$ and latency $t_{t,v}$) in low intensity environment of $|\mathcal{Z}_{t,v}| = 50$. With $|\mathcal{Z}_{t,v}| = 50$, we demonstrate how the proposed approach and baselines adapt to varying application latency requirements $\psi_{t,v}^{\text{lat}}$. As the latency requirement $\psi_{t,v}^{\text{lat}}$ should not be higher than the task generation period to avoid hindering task execution, this lower task generation rate allows us to explore a broader range of latency requirements $\psi_{t,v}^{\text{lat}}$. This approach provides insights into system performance across diverse operational scenarios. The outcomes of simulations are depicted in Fig. 8.

As depicted in Fig. 8a, the proposed approach as well as all benchmarks yield the accuracy $\mu_{t,v}$ safely above the required accuracy $\psi_{t,v}^{\text{acc}} = 0.82$ (see Table III). The accuracy reached by the proposal is marginally lower than the accuracy reached by *Edge AI* and *Server only*. Nevertheless, a slightly lower accuracy of the proposal compared to benchamrks is insignificant and meaningless, since the required accuracy is fulfilled. A small increase in accuracy $\mu_{t,v}$ is observed for all baselines, including the proposed approach, as latency requirement $\psi_{t,v}^{\text{lat}}$ increases. This is attributed to the selection of later exits $e_{t,v}$ across all baselines, coupled with the utilization of autoencoders $a_{s_{t,v},t,v}$ with lower compression within the proposed approach.

Fig. 8b shows, that the latency $t_{t,v}$ increases with the latency requirement $\psi_{t,v}^{\text{lat}}$. The latency for the proposal escalates at a lower rate in comparison to the *Edge ML* baseline, as such, the proposed approach is able to achieve latency $t_{t,v}$ improvement



(a) Accuracy $\mu_{t,v}$ under different amount of resource blocks available N^{RB}.

(b) Latency $t_{t,v}$ under different amount of resource blocks available N^{RB}.

(c) Dropped task ratio under different amount of resource blocks available N^{RB}.

Fig. 9: Performance evaluation at different levels of resource block availability N^{RB} with $W_v^{\mathrm{lat}} = 1$, $W_v^{\mathrm{acc}} = 0.15$, $\psi_{t,v}^{\mathrm{lat}} = 5.5$ms and $|\mathcal{Z}_{t,v}| = 180$.

of up to 36.67% relative to the baselines.

In the low workload intensity $|\mathcal{Z}_{t,v}| = 50$ and low latency requirement $\psi_{t,v}^{\mathrm{lat}}$, neither the baselines nor the proposed approach drop any task. Consequently, we present a graphs of the reward function $u_{t,v}$, as delineated in (15), within Fig. 8c. As depicted in Fig. 8c, all the baselines as well as the proposed approach exhibit a growth in the reward $u_{t,v}$ with increasing latency requirement $\psi_{t,v}^{\mathrm{lat}}$. This growth indicates that an increase in the latency requirement $\psi_{t,v}^{\mathrm{lat}}$ correlates with an increased satisfaction balance between the accuracy $\mu_{t,v}$ and the latency $t_{t,v}$. Notably, the proposed approach attains the highest reward $u_{t,v}$ at all points of the graph, achieving a reward $u_{t,v}$ that is up to 91.6% superior in comparison to the baselines.

As illustrated in all sub-figures Fig. 8a – Fig. 8c, the *Server only* approach maintains a consistent level of the accuracy $\mu_{t,v}$ and the latency $t_{t,v}$ across all investigated points. This uniformity is attributed to its invariance to changes in latency requirements $\psi_{t,v}^{\mathrm{lat}}$.

G. Communication Resource

Next, we create a high-intensity environment, where 180 tasks are generated per time interval t for all CAVs $v \in \mathcal{V}$. To this end, we intentionally increase the available data rate, as defined in (2), by varying the allocation of resource blocks N^{RB} for the CAVs. The rationale behind high-intensity environment choice lies in the need to stress-test the communication resources. In low-intensity environments, the demand for high communication resources might not be as critical. Results of the simulations are shown in Fig. 9.

As the number of resource blocks N^{RB} is increased, a reduction in latency is observed (see Fig. 9b), attributable to a decreased time required for the task offloading. Moreover, the integration of autoencoder selection within the proposed method leads to a significantly faster task offloading process and to a consequent latency reduction $t_{t,v}$ of up to 23.27%. It is noteworthy that the *Edge ML* and *Edge AI* baselines yield only marginal improvements in comparison to the *Server only* approach. This marginal enhancement is primarily due to baselines choosing to execute the entire tasks on the edge server, hence the improved latency is largely attributed to early exiting. This is because neither *Edge ML* nor *Edge AI* are able to find a splitting strategy that met the latency requirement $\psi_{t,v}^{\mathrm{lat}} = 5.5ms$, defaulting to full task offloading instead. This demonstrates the advantage of our proposed approach, which, through comprehensive weighing of accuracy $\psi_{t,v}^{\mathrm{acc}}$ and latency $\psi_{t,v}^{\mathrm{lat}}$, found an optimal solution despite a larger action space compared to *Edge ML* (as *Edge ML* does not select an appropriate autoencoder).

Additionally, an increase in the number of resource blocks N^{RB} correlates with a decrease in the ratio of dropped tasks, as shown in Fig. 9c. This, in turn, leads to an increment in the accuracy $\mu_{t,v}$, as depicted in Fig. 9a. The proposed method increases the accuracy $\mu_{t,v}$ of up to 36.6% compared to the baselines, while concurrently achieving a superior task drop rate reduction of up to 53.34%.

VI. Conclusion

In this paper, we have proposed a novel approach to joint strategy selection, encompassing determination of early exits, splits, and autoencoders for processing of the CNN-based computer vision tasks of CAVs. The proposed method leverages the MADDPG-based algorithm to identify the optimal strategy. Selection of the task processing strategy by each CAVs inevitably impacts the task processing of other CAVs. To address this interdependency, we introduce a cooperative multi-agent learning paradigm, where the CAVs implicitly incorporate the actions and states of other CAVs when learning to make strategic decisions. The cooperative approach enables the CAVs to maximize not only their own utility function, but also the utility of other CAVs. Simulation results demonstrate that the proposal lowers the latency compared the state-of-the-art baselines, while simultaneously minimizing the occurrence of dropped tasks in high-intensity, low-connectivity environments. Furthermore, we show that the proposed method maintains comparable accuracy and lower latency compared to state-of-the-art works even in low-intensity scenarios. The proposed approach also achieves highest values for the utility function, resulting in the most desirable accuracy-latency trade-off.

Our approach exhibits versatility and can be effectively implemented across a wide spectrum of CNN-based tasks. However, extending the proposed approach to explore other critical

computer vision tasks, such as semantic segmentation and object detection is a future challenge. Further research endeavors could also delve into deploying this algorithm within non-standard CNN architectures, including multi-head architectures, such as [68].

References

[1] S. Atakishiyev, M. Salameh, H. Yao, and R. Goebel, "Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions," *arXiv preprint arXiv:2112.11561*, 2021.

[2] S. Raza, S. Wang, M. Ahmed, M. R. Anwar, *et al.*, "A survey on vehicular edge computing: architecture, applications, technical issues, and future directions," *Wireless Communications and Mobile Computing*, vol. 2019, 2019.

[3] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, 2021.

[4] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, *et al.*, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[5] Ò. Lorente, I. Riera, and A. Rana, "Image classification with classic and deep learning techniques," *arXiv preprint arXiv:2105.04895*, 2021.

[6] H. Ajmal, S. Rehman, U. Farooq, Q. U. Ain, F. Riaz, and A. Hassan, "Convolutional neural network based image segmentation: a review," *Pattern Recognition and Tracking XXIX*, vol. 10649, pp. 191–203, 2018.

[7] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," *Advances in neural information processing systems*, vol. 26, 2013.

[8] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2020.

[9] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile networks and applications*, vol. 26, pp. 1145–1168, 2021.

[10] W. Fan, Y. Su, J. Liu, S. Li, W. Huang, F. Wu, and Y. Liu, "Joint task offloading and resource allocation for vehicular edge computing based on V2I and V2V modes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 4, pp. 4277–4292, 2023.

[11] S. Li, N. Zhang, H. Chen, S. Lin, O. A. Dobre, and H. Wang, "Joint road side units selection and resource allocation in vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13190–13204, 2021.

[12] S. Wang, J. Li, G. Wu, H. Chen, and S. Sun, "Joint optimization of task offloading and resource allocation based on differential privacy in vehicular edge computing," *IEEE Transactions on Computational Social Systems*, vol. 9, no. 1, pp. 109–119, 2021.

[13] Z. Xue, C. Liu, C. Liao, G. Han, and Z. Sheng, "Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems," *IEEE Transactions on Vehicular Technology*, 2023.

[14] X. Huang, L. He, and W. Zhang, "Vehicle speed aware computing task offloading and resource allocation based on multi-agent reinforcement learning in a vehicular edge computing network," in *2020 IEEE International Conference on Edge Computing (EDGE)*, pp. 1–8, IEEE, 2020.

[15] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, pp. 1–74, 2021.

[16] S. Chatterjee, P. Tummala, O. Speck, and A. Nürnberger, "Complex network for complex problems: A comparative study of CNN and complex-valued CNN," in *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)*, pp. 1–5, IEEE, 2022.

[17] K. Lu, R.-D. Li, M.-C. Li, and G.-R. Xu, "MADDPG-based joint optimization of task partitioning and computation resource allocation in mobile edge computing," *Neural Computing and Applications*, pp. 1–18, 2023.

[18] 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, 2020.

[19] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[20] A. Malawade, M. Odema, S. Lajeunesse-DeGroot, and M. A. Al Faruque, "Sage: A split-architecture methodology for efficient end-to-end autonomous vehicle control," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–22, 2021.

[21] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.

[22] S. Tuli, G. Casale, and N. R. Jennings, "SplitPlace: AI augmented splitting and placement of large-scale neural networks in mobile edge environments," *IEEE Transactions on Mobile Computing*, 2022.

[23] A. E. Eshratifar, A. Esmaili, and M. Pedram, "BottleNet: A deep learning architecture for intelligent mobile cloud computing services," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, 2019.

[24] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, "SC2 benchmark: Supervised compression for split computing," *Transactions on machine learning research*, 2023.

[25] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

[26] Z. Zhao, K. Wang, N. Ling, and G. Xing, "EdgeML: An autoML framework for real-time deep learning on the edge," IoTDI '21, (New York, NY, USA), p. 133–144, Association for Computing Machinery, 2021.

[27] S. Teerapittayanon, B. McDanel, and H. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016.

[28] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Why should we add early exits to neural networks?," *Cognitive Computation*, vol. 12, no. 5, pp. 954–966, 2020.

[29] A. K. Kosta, M. A. Anwar, P. Panda, A. Raychowdhury, and K. Roy, "RAPID-RL: A reconfigurable architecture with preemptive-exits for efficient deep-reinforcement learning," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 7492–7498, IEEE, 2022.

[30] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[31] N. J. Zakaria, M. I. Shapiai, R. A. Ghani, M. Yasin, M. Z. Ibrahim, and N. Wahid, "Lane detection in autonomous vehicles: A systematic review," *IEEE Access*, 2023.

[32] Y. Satılmış, F. Tufan, M. Şara, M. Karslı, S. Eken, and A. Sayar, "CNN based traffic sign recognition for mini autonomous vehicles," in *Information Systems Architecture and Technology: Proceedings of 39th International Conference on Information Systems Architecture and Technology–ISAT 2018: Part II*, pp. 85–94, Springer, 2019.

[33] W. Song, Y. Yang, M. Fu, F. Qiu, and M. Wang, "Real-time obstacles detection and status classification for collision warning in a vehicle active safety system," *IEEE Transactions on intelligent transportation systems*, vol. 19, no. 3, pp. 758–773, 2017.

[34] J. Chen, Y. Leng, and J. Huang, "An intelligent approach of task offloading for dependent services in mobile edge computing," *Journal of Cloud Computing*, vol. 12, no. 1, p. 107, 2023.

[35] 3GPP, "Nr; physical channels and modulation," *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.211*, vol. 9, 2018.

[36] V. Rajagopalan, Z. Jiang, J. Stojanovic-Radic, G. Yue, E. P. Pioro, G. Wylie, and A. Das, "A basic introduction to diffusion tensor imaging mathematics and image processing steps," *Brain Disord Ther*, vol. 6, no. 229, p. 2, 2017.

[37] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[38] D. Mukunoki and T. Imamura, "Reduced-precision floating-point formats on GPUs for high performance and energy efficient computation," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 144–145, 2016.

[39] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, pp. 1–6, Ieee, 2017.

[40] A. B. de Souza, P. A. L. Rego, T. Carneiro, P. H. G. Rocha, and J. N. de Souza, "A context-oriented framework for computation offloading

in vehicular edge computing using wave and 5G networks," *Vehicular Communications*, vol. 32, p. 100389, 2021.
[41] J. Zhou, D. Tian, Z. Sheng, X. Duan, and X. Shen, "Distributed task offloading optimization with queueing dynamics in multiagent mobile-edge computing networks," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12311–12328, 2021.
[42] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.
[43] A. Bozorgchenani, S. Maghsudi, D. Tarchi, and E. Hossain, "Computation offloading in heterogeneous vehicular edge networks: On-line and off-policy bandit solutions," *IEEE Transactions on Mobile Computing*, vol. 21, no. 12, pp. 4233–4248, 2021.
[44] L. Zhao, E. Zhang, S. Wan, A. Hawbani, A. Y. Al-Dubai, G. Min, and A. Y. Zomaya, "Meson: A mobility-aware dependent task offloading scheme for urban vehicular edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 4259–4272, 2023.
[45] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2092–2104, 2019.
[46] L. K. Muller, "Overparametrization of hypernetworks at fixed FLOP-count enables fast neural image enhancement," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 284–293, 2021.
[47] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, pp. 1–49, 2022.
[48] Z. Cheng, M. Min, M. Liwang, L. Huang, and Z. Gao, "Multiagent DDPG-based joint task partitioning and power control in fog computing networks," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 104–116, 2021.
[49] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
[50] M. Claesen and B. De Moor, "Hyperparameter search in machine learning," *arXiv preprint arXiv:1502.02127*, 2015.
[51] L. Chen, R. Jain, and H. Luo, "Learning infinite-horizon average-reward Markov decision process with constraints," in *International Conference on Machine Learning*, pp. 3246–3270, PMLR, 2022.
[52] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[53] Y. Bai, E. Yang, B. Han, Y. Yang, J. Li, Y. Mao, G. Niu, and T. Liu, "Understanding and improving early stopping for learning with noisy labels," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24392–24403, 2021.
[54] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.
[55] B. Lehle and J. Peinke, "Analyzing a stochastic process driven by ornstein-uhlenbeck noise," *Physical Review E*, vol. 97, no. 1, p. 012113, 2018.
[56] T.-H. Fan and Y. Wang, "Soft actor-critic with integer actions," in *2022 American Control Conference (ACC)*, pp. 2611–2616, 2022.
[57] A. Gao, T. Geng, S. X. Ng, and W. Liang, "A continuous policy learning approach for hybrid offloading in backscatter communication," *IEEE Communications Letters*, vol. 25, no. 2, pp. 523–527, 2021.
[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.
[59] P. Chaudhari, R. Achar, and S. Singh, "Enhancing lane recognition in autonomous vehicles using cross-layer refinement network," *IEEE Access*, 2024.
[60] M. Kondapally, K. N. Kumar, C. Vishnu, and C. K. Mohan, "Towards a transitional weather scene recognition approach for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2023.
[61] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.
[62] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "Benchmarking multi-agent deep reinforcement learning algorithms," 2020.
[63] D. K. Dewangan, S. P. Sahu, and K. V. Arya, "Vision-sensor enabled multi-layer CNN scheme and impact analysis of learning rate parameter for speed bump detection in autonomous vehicle system," *IEEE Sensors Letters*, 2024.
[64] Y.-J. Ku, S. Baidya, and S. Dey, "Adaptive computation partitioning and offloading in real-time sustainable vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13221–13237, 2021.
[65] S. S. Sarma, K. Sinha, G. Chakraborty, B. P. Sinha, *et al.*, "Optimal distribution of traffic in manhattan road networks for minimizing routing-time," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 6799–6820, 2020.
[66] Y. Saadi, S. E. Kafhali, A. Haqiq, and B. Nassereddine, "Simulation analysis of routing protocols using manhattan grid mobility model in manet," *arXiv preprint arXiv:1304.2035*, 2013.
[67] I. Mohamed, "Path-loss estimation for wireless cellular networks using Okumura/Hata model," *Science Journal of Circuits, Systems and Signal Processing*, vol. 7, no. 1, pp. 20–27, 2018.
[68] M. Canizo, I. Triguero, A. Conde, and E. Onieva, "Multi-head CNN–RNN for multi-time series anomaly detection: An industrial case study," *Neurocomputing*, vol. 363, pp. 246–260, 2019.

Robert Rauch is a PhD student at the Technical University of Košice (TUKE), where he earned his B.Sc. and M.Sc. degrees in 2019 and 2021, respectively. As a visiting researcher at the Czech Technical University in Prague in 2023, he expanded his expertise in computer vision, focusing on accelerating tasks for autonomous vehicles within the realm of vehicular edge computing. In 2024, he conducted research at the University of California, Irvine, further enhancing his experience in computer vision, autonomous vehicles, split computing, and sensor fusion.

Zdenek Becvar received M.Sc. and Ph.D. in Telecommunication Engineering from the Czech Technical University in Prague (CTU), Czech Republic in 2005 and 2010, respectively. Now, he is the Full Professor at the same university. From 2006 to 2007 and in 2009, he was with Sitronics R&D centre and with Vodafone R&D center at CTU, respectively. He was on internships at Budapest Politechnic, Hungary (2007), CEA-Leti, France (2013), and EURECOM, France (2016, 2019, 2023). He is recipient of the best paper award at European Wireless 2017, bronze medal at ACM Mobicom App contest 2015, and the Best Editor awards from IEEE Wireless Comm. Letters in 2024.

Pavel Mach received M.Sc. and Ph.D. in Telecommunication Engineering from the Czech Technical University in Prague, Czech Republic in 2006 and 2010, respectively. He is senior researcher in 5G mobile lab founded in 2015 at CTU in Prague focusing on 5G and beyond mobile networks. He has published 3 book chapters, more than 70 conference or journal papers and he is co-inventor of three U.S. patents.

Juraj Gazda is currently a Vice-Rector for Innovation and Technology Transfer and Professor with the Faculty of Electrical Engineering at the Technical University of Košice (TUKE), Slovakia. He has been a guest researcher at Ramon Llull University, Barcelona, and the Technical University of Hamburg-Harburg. He has been involved in development for Nokia Siemens Networks (NSN). In 2017, he was recognized as the Best Young Scientist at TUKE. Currently, he serves as the editor of KSII Transactions on Internet and Information Systems and as a guest editor of Wireless Communications and Mobile Computing (Wiley).