DISTRIBUTED DEEP REINFORCEMENT LEARNING VIA SPLIT COMPUTING FOR CONNECTED AUTONOMOUS VEHICLES

Robert RAUCH, Juraj GAZDA
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, Tel. +421 55 602 3175,
E-mail: {robert.rauch, juraj.gazda}@tuke.sk

ABSTRACT

This paper proposes the application of split computing paradigms for deep reinforcement learning through distributed computation between Connected Autonomous Vehicles (CAVs) and edge servers. While this approach has been explored in computer vision, it remains largely unexplored for reinforcement learning scenarios. We introduce a novel autoencoder trained directly through Deep Q-Network (DQN) rewards, wherein we optimize autoencoder layers using the DQN reward function while maintaining all other layers frozen. Our experimental results demonstrate that the proposed approach outperforms baseline methods by reducing data offloading requirements to the edge server by up to 98.7%. Additionally, this methodology not only decreases the data transmission burden but also achieves comparable rewards. In certain configurations, it even enhancing performance by up to 9.65%. The primary objective of this research is to reduce latency in deep reinforcement learning tasks for autonomous vehicles. In this regard, proposed approach achieves up to 66.5% improvement in latency reduction compared to baseline methods. These findings indicate that partial offloading through split computing offers significant benefits over both full offloading and complete on-device computation strategies for CAVs.

Keywords: *Connected autonomous vehicles, control theory, deep reinforcement learning, edge computing, split computing*

1. INTRODUCTION

Connected Autonomous Vehicles (CAVs) face increasing computational demands as they become more sophisticated [1]. As the industry continues to evolve rapidly, these vehicles require precise, real-time control to navigate dynamic environments safely [1]. However, current CAVs often lack sufficient onboard computational power to process the high volume of sensor data needed for real-time decision-making [1].

To address this limitation, it is becoming increasingly beneficial to offload computational tasks to nearby edge servers for processing. Extensive research has explored strategies for task offloading in terms of timing, location, and methodology [2]. However, many approaches do not specifically address CAV applications, which primarily rely on computer vision [3] and control theory [4]. Traditional offloading methods typically transfer entire tasks to edge servers, which underutilizes onboard computational resources while requiring high-bandwidth data transmission. Prior research has attempted to address these inefficiencies through partial task offloading approaches [5], where tasks suitable for parallelization can be distributed between the CAV and edge server. However, these approaches do not adequately account for the nature of CAV sensor systems, which predominantly rely on computer vision algorithms–particularly Deep Learning (DL) models. This creates a significant limitation. While DL models are capable of being parallelized, they typically require substantial cross-device data communication [6]. This intensive communication requirement makes practical parallelization between CAV and edge server largely inefficient in real-world deployments.

Split computing [7] offers a solution to this limitation by distributing the computational workload sequentially. In this paradigm, deep learning models begin initial processing on the CAV itself. Computation is then offloaded only after a specific neural network layer. This approach enables distribution of the computational load between CAV and edge server. Additionally, it allows for precise determination of computational allocation for each component. Previous research has investigated optimal offloading points based on task latency [7] (i.e., where data transfer is minimized, as offloading latency typically creates a bottleneck) and methods for dynamically adjusting split points based on environmental conditions [8]. These approaches, however, consider only computer vision tasks, which in CAVs are typically just one component within a larger pipeline for decision making and control [9]. Furthermore, most of these studies consider full data offloading (i.e., the entire output from a neural network layer is transferred to the server). This approach offers advantages over full offloading by improving throughput for edge servers [7]. This means that the edge server is able to serve more CAVs. However, the biggest disadvantage is the creation of bottlenecks in the offloading process due to the high volume of data that needs to be transferred. As such, most split points along the neural network aren't practical for offloading.

Current state-of-the-art research is therefore focusing on optimizing these split points by compressing data before offloading [10–12]. This is typically accomplished by inserting Convolutional Neural Network (CNN) architecture, specifically autoencoder architecture, at potential split points. Output of the layer is compressed on the CAV using encoder part of the autoencoder. This data is then offloaded to the server, where it is decoded using the decoder part of the autoencoder. These systems are trained by running the DL model, using the output of the neural network layer as both input to the autoencoder architecture and as ground truth, with the autoencoder trained through supervised learning (usually via restoration loss functions such as mean squared error) [13]. However, this compression is lossy and typically results in some model performance degradation.

Motivated by these prior works and the identified gap in evaluating split computing for CAV control applications, this research uses highway lane-changing scenarios [14] to assess split computing performance in reinforcement learning-based control tasks. We begin by developing a DL model capable of successfully navigating the chosen environment. We then incorporate autoencoder bottlenecks using a novel training approach. Specifically, we freeze the environment-solving DL model and rerun the training process. During this process, we apply gradient descent exclusively to the autoencoder layers. This methodology draws inspiration from techniques in training early exits, where new branches are integrated into neural networks and trained while maintaining frozen main neural network layers. This approach provides improved computational efficiency and preservation of the original model's performance characteristics, while still enabling multiple potential split points–facilitating possible dynamic adjustments based on changing conditions.

Main contributions of this paper can be summarized as follows:

- Evaluation of split computing for reinforcement learning-based control theory in the domain of CAVs.
- Novel approach to model splitting designed specifically for reinforcement learning-based scenarios.
- Extensive simulation-based evaluation demonstrating the strengths and performance benefits of the proposed approach.

The remainder of this paper is structured as follows. We begin by introducing the system model in Section 2. Next, we define our problem in Section 3. A detailed description of our proposed approach is provided in Section 4. We then evaluate the proposed approach in Section 5. Finally, we summarize our work and discuss future research directions in Section 6.

2. SYSTEM MODEL

In this section we introduce our system model under study. This includes both computational model to determine task latency for processing on CAVs and edge server and communication model to determine task latency for data offloading.

2.1. Task model

We consider N_V CAVs, where each CAV is defined as $v \in V$. Each CAV is connected to a single edge server b co-located at a base station. Note, that we only have a single edge server within our system model, similarly to [15]. This single-server model is appropriate as our research focuses specifically on optimizing the distribution of computational workload between the CAV and its serving edge server, making server selection considerations separate from and unrelated to our current investigation. Each CAV $v \in V$ generates N_{Z_t} tasks in time interval t. Each task $z \in Z_t$ contains the information required for calculating task execution latency (i.e., computational complexity for CAV $I_{z,v}$ and edge server $I_{z,b}$, data required to be offloaded $D_{z,s,c}$, where $s \in S$ is split point s from a set of possible split points S and $c \in C$ is compression size c from a set of possible compression sizes C). Tasks $z \in Z_t$ are generated periodically, simulating a scenario of a sensor on CAVs, such as a camera sensor.

2.2. Communication model

Here, we will define model used to calculate latency of task being offloaded to the edge server. We start by defining the connection data rate $d_{t,v}$ to edge server b for CAV $v \in V$. This is defined as follows [8]:

$$d_{t,v} = N_v^{\text{bit}} \rho_v \frac{N^{\text{RB}}}{N_V} R, \tag{1}$$

where N_v^{bit} represents the number of bits per symbol and ρ_v indicates the coding rate for transmissions from CAV v. Both N_v^{bit} and ρ_v are functions of the channel quality indicators and their corresponding modulation and coding scheme index, as determined in [16]. N^{RB} refers to the total resource blocks at the edge server's base station shared among N_V CAVs, while R defines the base station's symbol rate.

Now, we can define the offloading latency t_z^{offload} for task $z \in Z_t$ as follows:

$$t_z^{\text{offload}} = \frac{D_{z,s,c}}{d_{t,v}} + t_{z,w}^{\text{offload}}, \tag{2}$$

where $t_{z,w}^{\text{offload}}$ represents the waiting time before task $z \in Z_t$ is offloaded, accounting for queuing and other communication-related delays as described in [17].

2.3. Computational model

The computational model defines latency calculations for processing on both the CAV and edge server. This encompasses the computational complexity of the DL model up to the split point including the encoder portion of the autoencoder (which serves as an artificial bottleneck for compressing intermediate data), as well as the computational complexity of the decoder portion and remaining DL model layers.

The computational complexity on the CAV is defined as $I_{z,v}$ and on the edge server as $I_{z,b}$, both measured as the number of floating point operations to be executed. The resulting computational latency is a function of these complexity values and the computational capabilities of the CAV ν_v and edge server ν_b. This is formally defined as:

$$t_z^{\text{CAV}} = \frac{I_{z,v}}{\nu_v} + t_{z,w}^{\text{CAV}}, \tag{3}$$

$$t_z^{\text{edge}} = \frac{I_{z,b}}{\nu_b} + t_{z,w}^{\text{edge}}, \tag{4}$$

where $t_{z,w}^{\text{CAV}}$ and $t_{z,w}^{\text{edge}}$ represent the time task $z \in Z_t$ spends waiting in queue before processing on the CAV and edge server, respectively.

2.4. Total latency model

Finally, we define the total end-to-end latency for model execution. This comprises three components: the computational processing latency on the CAV t_z^{CAV}, the offloading latency for intermediate data transfer (output from the compressed layer) to the edge server t_z^{offload}, and the computational processing latency on the edge server t_z^{edge}. This can be formally expressed as:

$$t_z = t_z^{\text{CAV}} + t_z^{\text{offload}} + t_z^{\text{edge}}. \tag{5}$$

3. PROBLEM FORMULATION

In this section we define the objective of this study. First, we introduce supplementary equations that will help us formulate the problem more clearly. These supplementary equations define essential sets and temporal boundaries that govern task execution across the system. They are formally defined as follows:

$$Z_{t',v}^{\text{CAV}} = \{z | \tau_z \le t' \le \tau_z + t_z^{\text{CAV}}\}, \tag{6}$$
$$Z_{t',v}^{\text{offload}} = \{z | \tau_z + t_z^{\text{CAV}} \le t' \le \tau_z + t_z^{\text{CAV}} + t_z^{\text{offload}}\}, \tag{7}$$
$$Z_{t'}^{\text{edge}} = \{z | \tau_z + t_z^{\text{CAV}} + t_z^{\text{offload}} \le t' \le \tau_z + t_z\}, \tag{8}$$
$$\tau_z^{\text{CAV}} = \min(\tau_z + t_z^{\text{CAV}}, \tau_z + T_t), \tag{9}$$
$$\tau_z^{\text{offload}} = \min(\tau_z + t_z^{\text{CAV}} + t_z^{\text{offload}}, \tau_z + T_t), \tag{10}$$
$$\tau_z^{\text{edge}} = \min(\tau_z + t_z, \tau_z + T_t), \tag{11}$$

where $Z_{t',v}^{\text{CAV}}$, $Z_{t',v}^{\text{offload}}$ and $Z_{t'}^{\text{edge}}$ represent sets of active tasks at time point t' for CAV v (in the case of CAV processing and offloading) and edge server b (for edge processing) within the continuous timeline. τ_z denotes the generation time of task z. Meanwhile, τ_z^{CAV}, τ_z^{offload} and τ_z^{edge} define the temporal boundaries for each processing phase of the task, taking into account the potential deadline constraint T_t. These boundaries represent the adjusted completion times for CAV processing, data offloading, and edge server processing respectively, ensuring that task execution remains within the permissible time window.

Our objective is to minimize latency of task execution t_z by choosing appropriate split s and its compression size c:

$$\min_{\{s,c\}} \quad t_z \tag{12}$$
$$\text{s.t.} \quad s \in S, c \in C,$$
$$r > r',$$
$$t_z \le T_t, \forall z \in Z_t,$$
$$\sum_{z \in Z_{t',v}^{\text{CAV}}} \frac{I_{z,v,t'}}{v_v} \le 1, \forall t', \forall v \in V,$$
$$\sum_{z \in Z_{t',v}^{\text{offload}}} \frac{D_{z,s,c,t'}}{d_{t,v}} \le 1, \forall t', \forall v \in V,$$
$$\sum_{z \in Z_{t'}^{\text{edge}}} \frac{I_{z,b,t'}}{v_b} \le 1, \forall t',$$
$$\sum_{t'=\tau_z}^{\tau_z^{\text{CAV}}} I_{z,v,t'} \le I_{z,v}, \forall z \in Z_t, \forall v \in V,$$
$$\sum_{t'=\tau_z}^{\tau_z^{\text{offload}}} D_{z,s,c,t'} \le D_{z,s,c}, \forall z \in Z_t, \forall v \in V,$$
$$\sum_{t'=\tau_z+t_z^{\text{CAV}}}^{\tau_z^{\text{edge}}} I_{z,b,t'} \le I_{z,b}, \forall z \in Z_t,$$

where constraints 13 ensure that split point s and compression rate c are selected from predefined sets S and C. Constraint 13 ensures that the reward r (i.e., reward defined by the environment our agent is trying to solve–lane switching in our case) remains above a specified threshold r', guaranteeing that the quality of results stays acceptable despite lossy compression. Constraint 13 enforces that the execution latency t_z of task z does not exceed the deadline T_t defined for time interval t.

Constraints 13, 13, and 13 ensure that computational and communication resources are not overloaded at any moment. These constraints limit the instantaneous resource utilization at each processing stage, where $I_{z,v,t'}$ represents the instruction count executed on vehicle v, $D_{z,s,c,t'}$ represents the data volume offloaded, and $I_{z,b,t'}$ represents the instruction count executed on edge server b, all at time point t'.

Finally, constraints 13, 13, and 13 ensure that the total work performed for each task does not exceed its requirements. Specifically, they guarantee that the total floating-point operations executed on vehicle v are at most $I_{z,v}$, the total data offloaded does not exceed $D_{z,s,c}$, and the total floating-point operations executed on edge server b are at most $I_{z,b}$. These constraints allow for the possibility that tasks may not complete their full workload if deadline constraints intervene.

4. PROPOSED APPROACH

In this section, we present our proposed approach for addressing the problem formulated in Section 3. First, we describe the architecture and modifications of our deep learning model designed to handle the reinforcement learning (RL) environment in 4.1. Subsequently, we detail the algorithmic approaches developed to optimize the total latency t_z within the proposed framework in 4.2.

4.1. Offline configuration

Initially, we construct a DL architecture analogous to the structure depicted in Fig. 1, comprising an input layer (typically processing image data), followed by a sequence of convolutional layers, and culminating in fully connected (linear) layers that output predicted actions for subsequent environmental interaction. To optimize this architecture, we employ Deep Q-Network (DQN) training within reinforcement learning environment (lane switching in our case, but can be analogously used in any other task, where RL can be utilized). It is noteworthy that our proposed approach focuses exclusively on the neural network configuration, rendering it adaptable to alternative reinforcement learning algorithms.

Post-training, we strategically introduce split points as illustrated in Fig. 1, specifically positioning these junctures

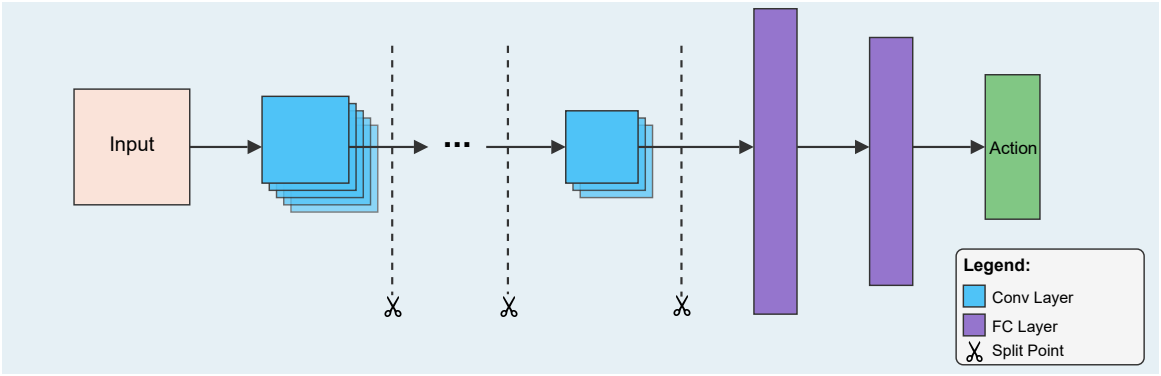


Fig. 1 Convolutional neural network with split points

after each convolutional layer. This selective placement is motivated by the observation that convolutional operations constitute the predominant computational burden, thus making splits between fully connected layers computationally inexpensive and therefore pointless, as there is negligible difference between placing a split before or within the fully connected layers. The computational complexity of convolutional layers scales with input dimensions, kernel sizes, and feature map counts, whereas fully connected layers exhibit more predictable computational profiles regardless of the network partition strategy.

Furthermore, we implement autoencoder-based compression mechanisms, which are structurally compatible with convolutional layers as they primarily utilize convolutional neural networks. This architectural compatibility facilitates seamless integration with the convolutional layers of our main network, whereas they would be incompatible with fully connected layer partitioning, further justifying our split point selection strategy. The integration of autoencoders at the identified split points enables effective dimensionality reduction of intermediate representations, which substantially reduces the communication overhead between distributed computational nodes.

4.2. Search for optimal split point and compression rate

As mentioned before, we are adding split points within our DL model (convolutional neural network). Each split point incorporates an autoencoder architecture that compresses data for offloading to the edge server. Here, we will first discuss how we choose the optimal compression rate, and then the optimal split point.

Algorithm 1 defines our proposed approach for determining optimal compression rates across all split points. Initially, we configure our DL model that solves the reinforcement learning environment (line 1). We then initialize the set of possible compression rates, an empty set that will store optimal compression rates for each split point (i.e., the number of values in C' should equal the number of split points S after running the algorithm), and establish a threshold for our reward function (lines 2-4).

The process begins by iterating through all split points S (line 5) and initializing the optimal compression rate c' for each split point (line 6). For each split point, we evaluate all possible compression rates C (line 7). To assess each compression rate, we first initialize an autoencoder with the specified compression rate (i.e., the compression rate corresponds to the number of channels in tensor we want to compress with the autoencoder), freeze all layers of the main DL model while keeping the autoencoder trainable, and then commence training our DL model (lines 8-10). It is important to note that the DL model does not update the weights of its main layers during this process; only the autoencoder layers are modified.

After training, we evaluate the effectiveness of the compression rate c by executing our environment and obtaining the average reward r (line 11). We then compare this reward against our predetermined threshold; if it exceeds the threshold, we designate it as the best compression rate c' and exit the compression rate loop (lines 12-15). Our objective is to identify the first compression rate that satisfies the reward threshold r' constraint. To achieve this efficiently, our compression rates set C must be ordered from highest compression to lowest compression, ensuring we attain maximum compression while meeting the reward threshold constraint.

We subsequently add the identified optimal compression c' to our collection of optimal compressions C' (line 17). It is worth noting that if no optimal compression c' is found, a null value is added to the set. This becomes problematic only if all values in the entire set of optimal compression rates C' are null. Such an outcome would indicate that no viable solution exists for the current configuration with the specified reward threshold– essentially signifying that none of the compression rates at any split point can satisfy our performance requirements. Finally, we return this set of optimal compression rates C' (line 19).

Algorithm 1 Search for optimal compression rate

1: Configure the DL model with S split points
2: Initialize possible compression rates C
3: Initialize empty set C' of optimal compression rates
4: Initialize threshold for reward r'
5: **for** $s \in S$ **do**
6: $c' \leftarrow$ null
7: **for** $c \in C$ **do**
8: Initialize autoencoder with $c \in C$ compression rate and insert it into the DL model
9: Freeze all layers except the autoencoder
10: Train DL model with the autoencoder using RL
11: Evaluate the model and get average reward r
12: **if** $r > r'$ **then**
13: $c' \leftarrow c$
14: **break**
15: **end if**
16: **end for**
17: $C' \leftarrow c'$
18: **end for**
19: return set of optimal compression rates C'

Algorithm 2 Search for optimal split point

1: Initialize DL model with split points S and optimal compression rates C' for each split
2: Initialize optimal split $s' \leftarrow$ null
3: Initialize best latency $t_z^{\text{best}} \leftarrow \infty$
4: **for** $s \in S$ **do**
5: Initialize autoencoder with best compression rate c' from C' for split s
6: Run edge computing environment for n steps, where we offload according to the split s and compression c'
7: Measure average latency t_z^{avg}
8: **if** $t_z^{\text{avg}} < t_z^{\text{best}}$ **then**
9: $t_z^{\text{best}} \leftarrow t_z^{\text{avg}}$
10: $s' \leftarrow s$
11: **end if**
12: **end for**
13: **return** the optimal split point s'

Now that we have determined a set of optimal compression rates C', we proceed to identify the optimal split point. Algorithm 2 outlines this process. We begin by initializing our DL model, establishing a null value for the optimal split s', and setting an initial value for the best latency t_z^{best} (lines 1-3).

The algorithm then systematically evaluates each split point in the set S (line 4). For each split point, we initialize an autoencoder with the corresponding optimal compression rate c' from the set of optimal compressions C' previously determined by Algorithm 1. Subsequently, we execute our edge computing environment simulation with CAVs, offloading the DL model according to the current configuration (i.e., current split point and compression rate). During this simulation, we measure task latency t_z and calculate its average value t_z^{avg} (lines 6 and 7).

If the measured average latency t_z^{avg} is lower (and therefore better) than our previously recorded best latency t_z^{best}, we update our best latency record to the current value t_z^{avg} and designate the current split point s as our optimal split point s' (lines 8-11). This iterative comparison ensures that we identify the split point that minimizes latency while maintaining the performance constraints established during compression rate optimization.

5. RESULTS

In this section, we present a comprehensive evaluation of our proposed approach. We begin with a description of the simulation environment, followed by an introduction to the baseline methods used for comparative analysis. Subsequently, we analyze the performance of our autoencoder compression technique and evaluate the efficacy of different computational splits across various network scenarios.

5.1. Simulation setup

Table 1 Simulation Parameters

Number of CAVs (N_v)	10
Task generation rate	50 tasks/second
Resource blocks (N^{RB})	1250
Base station symbol rate (R)	2×10^9 symbols/s
Path loss model	Hata model
Channel fading	Rayleigh fading
Base station computation power (ν_b)	2.6×10^{12} FLOPS
Vehicle computation power (ν_v)	50×10^9 FLOPS
Simulation runs	10
Time steps per run	5000

We conducted extensive simulations to evaluate the performance of our proposed framework. Table 1 summarizes the key simulation parameters. The network consists of $N_v = 10$ CAVs, each generating 50 computational tasks per second at regular intervals to simulate the operation of vehicular camera sensors with fixed frame rates. Our default configuration employs $N^{\text{RB}} = 1250$ resource blocks. The base station operates at a symbol rate of $R = 2 \times 10^9$ symbols per second. For channel modeling, we implement the Hata path loss [18] model combined with Rayleigh channel fading [19] to accurately represent urban wireless environments. The base station's computational capacity is set to $\nu_b = 2.6 \times 10^{12}$ floating point operations per second (FLOPS), while each vehicle possesses $\nu_v = 50 \times 10^9$ FLOPS. Our 10 CAVs follow the Manhattan mobility model, similarly to [20], on a Manhattan-like grid urban environment. This creates a non-stationary environment, representing realistic dynamic traffic within a city. As vehicles move through the grid, they experience continuously changing positions and network conditions, combined with Hata path loss

and Rayleigh channel fading, thus closely mimicking real-world autonomous driving scenarios.

All performance metrics are averaged over 10 independent simulation runs, each consisting of 5000 time steps, to ensure statistical significance and account for the stochastic nature of the wireless channel and vehicle mobility.

The reinforcement learning environment for our CAVs is a lane-switching scenario implemented using the Highway gym environment [21]. This environment provides a bird's-eye view of vehicles on a multi-lane road, where the primary objective is to navigate between lanes efficiently to overtake other vehicles while maintaining safety constraints. Bird's-eye view representations are particularly relevant as they are currently an active area of research in autonomous driving [22] and are widely deployed in real-world autonomous navigation systems for similar decision-making tasks.

5.2. Baselines

To evaluate the efficacy of our proposed approach, we compare it against two principal baselines that represent the extremes of the computation offloading spectrum in vehicular edge computing:

- *CAV Only:* This baseline executes the entire deep learning inference task locally on the CAV without any computation offloading. This approach is expected to perform adequately when vehicles possess sufficient computational resources, but may introduce significant latency when processing complex neural network models.

- *Edge Only:* In this baseline, the complete task is offloaded to the edge server. The vehicle transmits the entire sensor data to the edge infrastructure, which executes the deep learning model and returns only the resulting action values. While this approach leverages the superior computational capabilities of edge servers, it may suffer from communication overhead, especially in scenarios with limited bandwidth or high network congestion.

These baselines provide meaningful performance bounds to demonstrate how our proposed distributed computational splitting approach can optimize real-time execution for deep reinforcement learning tasks for autonomous driving.

5.3. Compression search

Fig. 2 illustrates our compression search strategy across all splits. For each split, we begin with compression $c = 1$ (compressing tensor data to a single channel before offloading and later restoring to the original channel count). We incrementally increase c until we achieve a reward r exceeding our threshold $r' = 65$. Interestingly, we observe fluctuations in reward at certain compression sizes, most notably at $c = 2$ and particularly for split point 3. This can be attributed to the sensitivity of deep learning models to architectural changes—as compression size alters the architecture, certain configurations perform better than others. Nevertheless, we observe the expected general upward trend in rewards as compression size c increases.

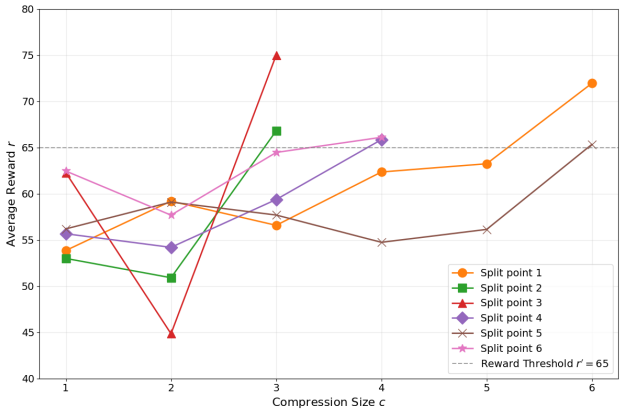


Fig. 2 Results of compression c search across different split points

Fig. 3 shows the results for different strategies with chosen compression rate c. We show the data size $D_{z,s,c}$ which needs to be offloaded for different strategies and achieved reward r across different strategies. We can see, that even with high compressions, our rewards are around the *Edge Only* and *CAV Only* approaches, even outperforming them for split 3 by up to 9.65% compared to the baselines. Furthermore, split 6 only needs 0.01 MB of data to offload, which is a 98.7% improvement over the *Edge Only* baseline.

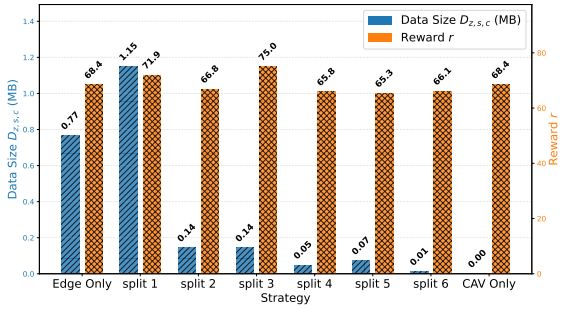
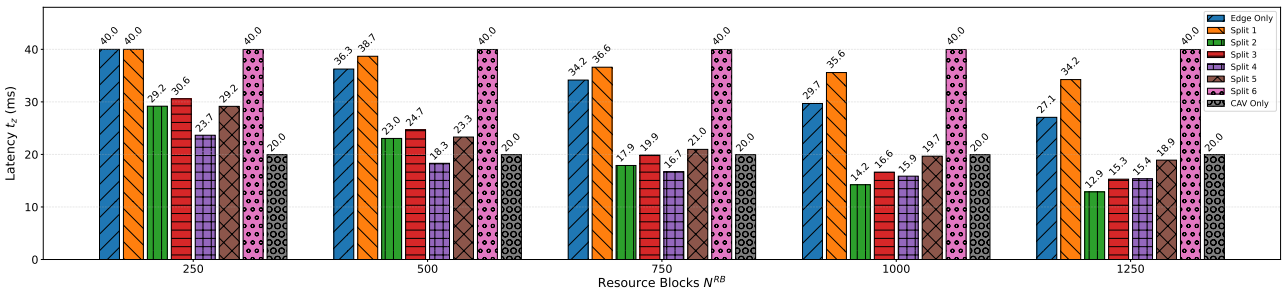


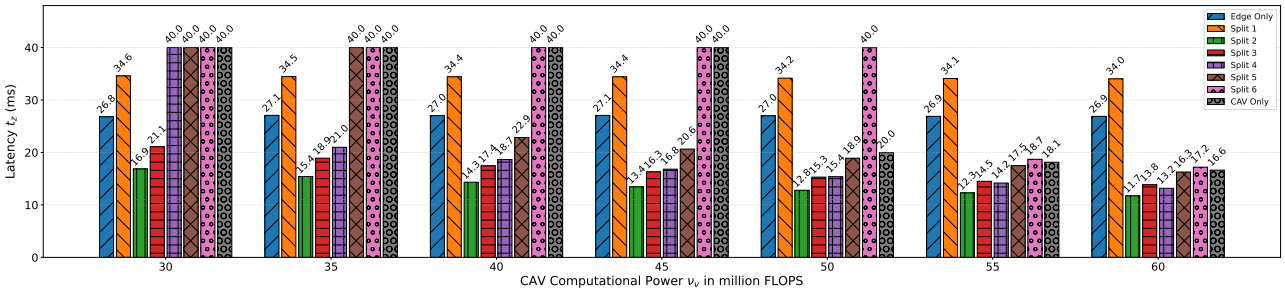
Fig. 3 Data size and reward across different strategies

5.4. Split search

To determine the optimal split strategy, we conducted extensive simulations across diverse parameter settings and computed the mean latency for each strategy. Subsequently, we employed algorithm 2 to systematically identify the optimal partitioning strategy. Fig. 5 illustrates these experimental outcomes, with a star indicating the strategy that our algorithm identified as optimal. The results demonstrate that split 2 consistently outperformed baselines, yielding a 45.5% reduction in latency. This substantial performance improvement validates the efficacy of our proposed approach.



(a) Effect of different resource blocks for each split point in comparison with *Edge Only* and *CAV Only* baselines



(b) Effect of different CAV computational power ν_v for each split point in comparison with *Edge Only* and *CAV Only* baselines

Fig. 4 Comparison of proposed approach against *Edge Only* and *CAV Only* baselines in different scenarios

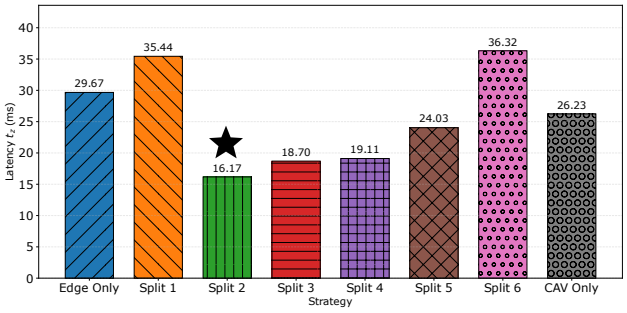


Fig. 5 Finding optimal split point

To evaluate our chosen split point, we conducted comprehensive simulations across diverse scenarios, as illustrated in Fig. 4. We systematically varied key parameters to assess performance robustness. First, we examined the impact of resource block allocation by incrementally increasing N^{RB} from 250 to 1250 in increments of 250, with results presented in Fig. 4a. Subsequently, we investigated the effect of CAV computational capacity by varying ν_v from 30 to 60 million FLOPS in 5 million FLOPS increments, as shown in Fig. 4b.

For comprehensive analysis, we included all alternative split configurations and baseline approaches. Our results reveal that when resource blocks N^{RB} are limited, split point 2 does not perform optimally compared to split 4, attributable to the higher data offloading requirements of split point 2 (as evidenced in Fig. 3). In low communications resource scenarios, the *CAV Only* approach demonstrates superior performance–an expected outcome since limited data transmission rates to the edge server can impede offloading efficiency. However, as N^{RB} increases, split 2 emerges as the optimal configuration, achieving up to 52.4% performance improvement compared to baseline approaches.

In our second experimental scenario, where we modulated CAV computational power ν_v, we observed that at lower computational capacities, later splits and the *CAV Only* approach reach the task deadline T_t of 40ms. As computational resources increase, more split configurations become viable; nevertheless, split 2–identified as optimal by our algorithm–consistently outperforms all baseline configurations across the parameter space. Notably, our proposed approach demonstrates remarkable efficiency, achieving up to 66.5% performance enhancement relative to baselines.

6. CONCLUSION

In this paper, we investigated the application of split computing paradigms for deep learning models within reinforcement learning environments. We selected lane-switching navigation as our reinforcement learning task, which CAVs must solve efficiently. To optimize the DL model architecture, we strategically incorporated autoencoders to compress data prior to transmission to the edge server. Notably, rather than employing conventional loss functions such as Mean Squared Error for autoencoder training, we implemented an innovative approach–freezing all other network layers and exclusively training the autoencoder components using reinforcement learning rewards through the DQN training algorithm.

Following comprehensive evaluation of these reinforcement learning-optimized autoencoders, we

systematically assessed all potential split configurations and identified the optimal partitioning strategy. We then rigorously benchmarked this configuration against both *Edge Only* and *CAV Only* baseline approaches. Our experimental results demonstrate that in the majority of scenarios with adequate connectivity to the edge server, split computing architectures significantly outperform conventional approaches, with our optimally selected split point consistently delivering superior performance compared to established baselines. These findings underscore the efficacy of reinforcement learning-guided split computing for latency-sensitive CAV applications.

However, as demonstrated in our experimental results, there exist specific scenarios where the *CAV Only* approach or alternative split points outperform our defined optimal split. This observation suggests that rather than designating a single split point as universally optimal, a more adaptive approach would be beneficial–one that implements multiple potential split configurations, including full offloading and complete on-device computation, while dynamically determining the optimal partitioning strategy based on prevailing environmental conditions. While such dynamic partitioning has been investigated in other domains, it remains largely unexplored within reinforcement learning scenarios and represents a promising direction for future enhancement of our proposed approach.

Although our study focuses on a single architecture, the results offer significant insights into the fundamental mechanisms of split computing for DRL. This targeted approach enabled us to conduct an in-depth analysis of layer-specific effects. Future research should extend this work by evaluating our proposed methods across diverse architectures and environments to further validate the broader applicability of our findings.

ACKNOWLEDGEMENT

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

Icons used in this work were made by Freepik from flaticon.com.

Part of the Research results were 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.

REFERENCES

[1] MA, Y. – WANG, Z. – YANG, H. – YANG, L.: Artificial intelligence applications in the development of autonomous vehicles: A survey. IEEE/CAA Journal of Automatica Sinica, 7(2), 2020, pp. 315–329

[2] YAN, G. – LIU, K. – LIU, C. – ZHANG, J.: Edge intelligence for internet of vehicles: A survey. IEEE Transactions on Consumer Electronics, 2024

[3] PAREKH, D. – PODDAR, N. – RAJPURKAR, A. – CHAHAL, M. – KUMAR, N. – JOSHI, G. P. – CHO, W.: A review on autonomous vehicles: Progress, methods and challenges. Electronics, 11(14), 2022, pp. 2162

[4] HUNG, N. – REGO, F. – QUINTAS, J. – CRUZ, J. – JACINTO, M. – SOUTO, D., et al.: A review of path following control strategies for autonomous robotic vehicles: Theory, simulations, and experiments. Journal of Field Robotics, 40(3), 2023, pp. 747–779

[5] REN, C. – ZHANG, G. – GU, X. – LI, Y.: Computing offloading in vehicular edge computing networks: Full or partial offloading?. In: 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC), Vol. 6, 2022, pp. 693–698. IEEE

[6] CHOI, H. – LEE, B. H. – CHUN, S. Y. – LEE, J.: Towards accelerating model parallelism in distributed deep learning systems. Plos one, 18(11), 2023, pp. e0293338

[7] KANG, Y. – HAUSWALD, J. – GAO, C. – ROVINSKI, A. – MUDGE, T. – MARS, J. – TANG, L.: Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. ACM SIGARCH Computer Architecture News, 45(1), 2017, pp. 615–629

[8] RAUCH, R. – BECVAR, Z. – MACH, P. – GAZDA, J.: Cooperative Multi-Agent Deep Reinforcement Learning for Dynamic Task Execution and Resource Allocation in Vehicular Edge Computing. IEEE Transactions on Vehicular Technology, 2024

[9] YOU, C. – LU, J. – FILEV, D.– TSIOTRAS, P.: Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. Robotics and Autonomous Systems, 114, 2019, pp. 1–18

[10] MATSUBARA, Y. – YANG, R. – LEVORATO, M.– MANDT, S.: SC2 benchmark: Supervised compression for split computing. arXiv preprint arXiv:2203.08875, 2022

[11] GUERRERO-BALAGUERA, J. D. – CONDIA, J. E. R. – LEVORATO, M. – REORDA, M. S.: Evaluating the Reliability of Supervised Compression for Split Computing. In: 2024 IEEE 42nd VLSI Test Symposium (VTS), 2024, pp. 1–6. IEEE

[12] MATSUBARA, Y. – CALLEGARO, D. – SINGH, S. – LEVORATO, M. – RESTUCCIA, F.: Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing. In: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2022, pp. 337–346. IEEE

[13] MATSUBARA, Y. – LEVORATO, M. – RESTUCCIA, F.: Split computing and early exiting for deep learning applications: Survey and research

challenges. ACM Computing Surveys, 55(5), 2022, pp. 1–30

[14] ATOUI, H. – SENAME, O. – MILANÉS, V. – MARTINEZ, J. J.: Intelligent control switching for autonomous vehicles based on reinforcement learning. In: 2022 IEEE Intelligent Vehicles Symposium (IV), 2022, pp. 792–797. IEEE

[15] ZHAO, Z. – WANG, K. – LING, N. – XING, G.: EdgeML: An AutoML framework for real-time deep learning on the edge. In: Proceedings of the International Conference on Internet-of-Things Design and Implementation, 2021, pp. 133–144.

[16] 3GPP: NR; Physical channels and modulation. 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.211, Vol. 9, 2018

[17] DE SOUZA, A. B. – REGO, P. A. L. – CARNEIRO, T. – ROCHA, P. H. G. – DE SOUZA, J. N.: A context-oriented framework for computation offloading in vehicular edge computing using WAVE and 5G networks. Vehicular Communications, 32, 2021, pp. 100389. Elsevier

[18] ALSHAMI, M., et al.: Evaluation of path loss models at WiMAX cell-edge. In: 2011 4th IFIP International Conference on New Technologies, Mobility and Security, 2011. IEEE.

[19] LIU, G., et al.: Deep learning-based channel prediction for edge computing networks toward intelligent connected vehicles. In: IEEE Access, vol. 7, 2019, pp. 114487–114495.

[20] LI, M. – SI, P. – ZHANG, Y.: Delay-tolerant data traffic to software-defined vehicular networks with mobile edge computing in smart city. In: IEEE Transactions on Vehicular Technology, vol. 67, no. 10, 2018, pp. 9073–9086.

[21] LEURENT, E.: An Environment for Autonomous Driving Decision-Making. GitHub repository, 2018 `https://github.com/eleurent/highway-env`

[22] DU, J. – SU, S. – FAN, R. – CHEN, Q.: Bird's eye view perception for autonomous driving. In: Autonomous Driving Perception: Fundamentals and Applications. Springer, 2023, pp. 323–356

Received April 8, 2025, accepted May 19, 2025

BIOGRAPHIES

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.

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).