SLO-Aware Task Offloading Within Collaborative Vehicle Platoons

Boris Sedlak[1](✉), Andrea Morichetta[1], Yuhao Wang[2], Yang Fei[2], Liang Wang[2], Schahram Dustdar[1], and Xiaobo Qu[2]

[1] Distributed Systems Group, TU Wien, Vienna, Austria
b.sedlak@dsg.tuwien.ac.at
[2] Emerging Transportation Solutions, Tsinghua University, Beijing, China

Abstract. In the context of autonomous vehicles (AVs), offloading is essential for guaranteeing the execution of perception tasks, e.g., mobile mapping or object detection. While existing work on offloading focused extensively on minimizing inter-vehicle networking latency, vehicle platoons (e.g., heavy-duty transport) present numerous other objectives, such as energy efficiency or data quality. To optimize these Service Level Objectives (SLOs) during operation, this work presents a purely Vehicle-to-Vehicle approach (V2V) for collaborative services offloading within a vehicle platoon. By training and using a Bayesian Network (BN), services can proactively decide to offload whenever this promises to improve platoon-wide SLO fulfillment; therefore, vehicles estimate how both sides would be impacted by offloading a service. In particular, this considers resource heterogeneity within the platoon to avoid overloading more restricted devices. We evaluate our approach in a physical setup, where vehicles in a platoon continuously (i.e., every 500 ms) interpret the SLOs of three perception services. Our probabilistic, predictive method shows promising results in handling large AV platoons; within seconds, it detects and resolves SLO violations through offloading.

Keywords: Service Level Objectives · Edge Computing · Intelligent Transportation · Microservices · Offloading · Bayesian Networks

1 Introduction

The swift evolution of Autonomous Vehicles (AVs) promises a disruptive impact [18] for future transportation. Despite AV solutions claim considerable benefits, such as rapid green transition and traffic flow improvement [14] , the execution of AV-enabling services, such as perception, path planning, and control [15] pose ambitious processing requirements. Here, optimal allocation and

Partially funded by the National Natural Science Foundation of China (NSFC) under grant numbers 52220105001 and 72322002, and partially by the European Union under (TEADAL, 101070186) and (INTEND, 101135576).

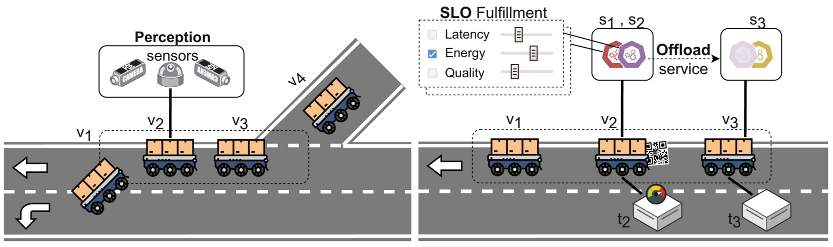
execution of workloads highly depend on AVs' constrained computation capabilities and the supporting infrastructure's network bandwidth. A lack of these guarantees can cause delays in real-time perception and decision-making, leading to potentially harmful consequences.

Services offloading [7] aims at mitigating these risks, for example, by minimizing computation latency between neighboring vehicles through Vehicle-to-Vehicle (V2V) or Vehicle-to-Infrastructure (V2I) transmission. However, collaborative AV scenarios commonly have higher-level objectives besides latency. For instance, consider AV platoons for public or heavy-duty transport, where the system providers want to minimize costs or energy consumption. We define these requirements as Service Level Objectives (SLOs) – a term from software engineering. The concept of SLOs is wide enough to define any high- or low-level objective that a management framework can enforce [19,23] by elastically adapting hardware or software. SLO-awareness also offers promising scenarios [22] for V2V offloading; however, its adoption remains limited, highlighting the gap for more intelligent offloading mechanisms [10].

This work, therefore, aims to ensure SLOs by incorporating them into the offloading mechanism – we call this "SLO-aware task offloading". Our motivation stems from two central objectives: (1) we want to ensure that vehicles fulfill the SLOs of their local services; if SLOs are violated, this might be resolved by offloading services, and simultaneously, (2) offloaded tasks must not jeopardize the SLO fulfillment of existing services at the target host. This goal implies solving a combinatorial problem, i.e., the optimal assignment of n services to m vehicles; this problem is NP-hard, hence practically intractable. A solution could be to decompose the problem so that AVs make decentralized offloading decisions. However, training an offloading model for every AV separately would introduce a considerable overhead. Furthermore, we would miss the chance to combine knowledge from multiple AVs, which promises a more profound understanding. For these reasons, we envision a method that trains a decision model within an AV but simultaneously integrates knowledge from other AVs.

In this paper, we present a modular, collaborative framework for autonomous SLO interpretation and service offloading. Here, we consider collaborative offloading approaches using "decentralized" sensory data [9]. Individual services continuously observe their processing to understand the extent to which SLOs can be fulfilled on different processing hardware; this knowledge is encoded in an SLO interpretation (SLO-I) model. These models are updated by a mutable platoon leader according to AVs' observations and then broadcast to other AVs. Given the SLO-I model, individual services predict how offloading would impact global SLO fulfillment. Hence, the contributions of this article are:

1. An SLO-aware offloading mechanism based on Bayesian networks that dynamically estimates the hardware implications of multiple competing services to find a satisfying assignment. Thus, it is possible to optimize the SLO fulfillment by shifting computation within a composable vehicle platoon.
2. A collaborative training strategy that continuously exchanges model updates between edge devices while adjusting the training frequency according to agents' local SLO prediction errors. Thus, service agents improve their SLO interpretation whenever the system does not behave as predicted.



(a) Maneuvers of the platoon (b) Offloading within the platoon

Fig. 1. Composite vehicle platoons offload computations according to SLO fulfillment; if service s_2's SLOs are not fulfilled at host v_2, it searches for alternatives, such as v_3

3. A modular framework for collaborative service offloading that can be extended with custom processing services and respective SLOs. Thus, other service managers can plug their own service implementation into the framework, which itself can be installed on arbitrary edge device types.

The remainder of this paper is organized as follows: Sect. 2 further illustrates the scenario used throughout this paper and gives an overview of related work, Sect. 3 describes our framework for SLO-aware offloading, which is evaluated in Sect. 4. Finally, we summarize our paper in Sect. 5.

2 Preliminaries

This section highlights the gap addressed by our methodology by presenting an illustrative scenario with vehicle platoons, where SLO awareness is essential for meeting high-level requirements, and by reviewing how this issue has been approached in existing research.

2.1 Illustrative Scenario

Here, we consider a platoon of vehicles for heavy-duty transportation. Depending on the trajectories of platoon members, individual vehicles can join or leave the platoon at specific intersections, such as ramps. One of the platoon members is elected as the leader, either apriori or dynamically. In our work, we focus on V2V offloading, as V2I infrastructures could be impractical [7] or add delays [3].

As shown in Fig. 1a, n vehicles are clustered into a platoon $P = \{v_1, ..., v_n\}$. We represent each vehicle through the pair $v = \langle id, t \rangle$, where $v.t$ specifies the type of processing device embedded. Additionally, each vehicle is equipped with numerous sensors and perception services, for instance, in Fig. 1b, vehicle v_2 runs two services, i.e., mapping its surroundings through Lidar (s_1) and detecting objects on the road through computer vision (s_2). Given that v_2 has a QR code attached to its rear, v_3 follows its predecessor by scanning for QR codes (s_3). We define a service through $s = \langle type, Q, C \rangle$, which reflects the *type* of perception service, e.g., Lidar or CV; Q specifies a set of processing SLOs, and C a list of service constraints, e.g., CV should operate at $fps = 15$. These specifications ensure safe operations when vehicles must respond to dynamic conditions.

Depending on services' resource demand, vehicles may not possess sufficient processing capabilities to fulfill their SLOs, which impacts the latency and quality of how a vehicle perceives its environment. For instance, v_2 might employ a weaker processing device ($v_2.t$); however, v_3's resources are less utilized, so v_2 might offload one of its services to v_3. Therefore, v_1 must now decide (1) which service, i.e., s_1 or s_2, should best be offloaded to v_3, (2) whether this improves SLO fulfillment of remaining services at v_2, and (3) if offloading could impact s_3 negatively. In the context of this paper, we focus on higher-level requirements, i.e., leaving out networking latencies for transferring input data and results under the assumption of high network throughput between nearby vehicles.

2.2 Related Work

We classify existing literature on task offloading for IoV and related scenarios in two main categories: offloading in V2I/V2V scenarios and offloading through Markovian or Bayesian methods. To set the foundation for our contribution, we highlight the strengths and limitations of these approaches.

IoV Offloading Mechanisms. In the context of V2I task offloading, Xu et al. [28] provide a neighborhood search algorithm that minimizes costs of task outsourcing, estimated on simulated network traffic. Similarly, Dong et al. [4] provide a multi-task and multi-user offloading mechanism for Mobile Edge Computing (MEC), optimized through a particle swarm. Ant colony optimization (ACO) is another explorative algorithm for optimal pathfinding: Mousa and Hussein [20] apply ACO to cluster IoT devices accessed by UAVs; Ma et al. [17] model the same scenario, but with Mixed-Integer Linear Programming (MILP), closely to Zhang et al. [29]. Related to our use case, Lu et al. [16] provide a latency-aware V2V/V2I offloading mechanism based on Deep Reinforcement Learning (DRL). Fan et al. [7] propose a V2V/V2I offloading tool that decomposes optimization problems with Generalized Benders Decomposition (GBD).

Other authors model offloading scenarios as shortest path [8] or stochastic optimization problem [13]; some methodologies focus on **solely V2V** offloading: Du et al. [5] provide a collaborative offloading mechanism for sensing tasks in autonomous vehicle platoons, making use of idle resources. Guo et al. [11] combine LSTM-based trajectory prediction and optimization strategy for V2V offloading. However, all these methods, while solid, rely on simulations rather than real-world data, assume static and homogeneous infrastructures, which are unrealistic, and frequently neglect SLO measures like energy consumption.

Offloading Through Markovian and Bayesian Methods. To the best of our knowledge, there are no solutions based on Bayesian Networks for V2V task offloading in platoons. Still, Markov models and Bayesian approaches are found in Edge-to-Cloud scenarios for task offloading [23,24]. Hazra et al. [12] use MILP to find offloading locations in hierarchical computing environments under latency and energy constraints. Wu et al. [27] offload streaming tasks from edge nodes to fog or cloud resources through a Markov decision process, improved through Reinforcement Learning (RL). Tasoulas et al. [25] provide a prediction mechanism that uses historical observations to forecast VMs' resource demand through Bayesian Networks. However, these papers offer little variety for SLOs and do not incorporate dynamic or real-time adaptations.

Takeaways. Existing research focused extensively on MEC offloading mechanisms to RSUs or UAVs for optimizing network latency; however other objectives, as energy efficiency or QoS are often overlooked. In addition, most approaches were only evaluated in simulations; however, to establish reliable offloading mechanisms, it is paramount to consider dynamic runtime behavior. Conversely, we propose an SLO-aware mechanism for V2V offloading that optimizes various SLOs in heterogeneous vehicle platoons. Centralized approaches suffer from the combinatorial complexity of finding a global optimum and the risk of becoming a single point of failure; in our approach, however, services have decentralized authority to interpret their runtime behavior and make offloading decisions.

3 Methodology

In the following, we present our modular framework for SLO-aware task offloading in composable vehicle platoons. This means, continuously observing service executions to collect insights, interpreting these insights through collaborative training, and making offloading decisions. Figure 2 provides a high-level overview of these processes, which are explained in more detail in Subsects. 3.1 to 3.3.

3.1 Service Observation

The first building block of our approach is observing a service, i.e., continuously monitoring and interpreting its SLO fulfillment. Observation requires interpreting service metrics parallel to service execution, as part of the service wrapper in Fig. 2. Perception tasks, such as those executed by autonomous vehicles, usually work iteratively; hence, service metrics are also interpreted step by step. In Algorithm 1, it is depicted how metrics ($D_{s,v}$) from executing a service (s) on a vehicle (v) are interpreted: for a set of SLOs (Q), the percentage of metrics (ϕ)[1] that fulfill these conditions is determined as shown in Eq. (1); then, ϕ is appended to the sliding window W_ϕ. To avoid overhasty decisions based on sporadic SLO violations, the length of the sliding window ($|W_\phi|$) can be customized.

$$\phi(Q) = \frac{\sum_{i=1}^{|Q|} \phi(q_i)}{|Q|} \tag{1a}$$

[1] We choose the symbol ϕ due to the sound of the letter, i.e., SLO ful-phi-llment.

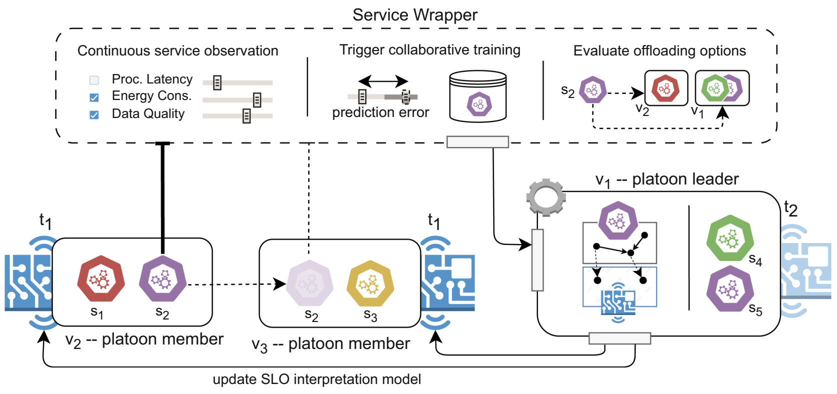


Fig. 2. Framework for collaborative offloading: inaccurate SLO predictions trigger retraining of SLO interpretation models; services use these models to evaluate alternative hosts according to their expected hardware utilization and SLO fulfillment

$$\phi(q_i) = \phi(q_i, m, v | \forall m \in D^v_{q_i}, v \in V) = \sum_{j=1}^{|D^v_{q_i}|} \frac{\phi(m_j, q_i)}{|D^v|} \tag{1b}$$

$$\text{where } \phi(q_i, m_j) = \begin{cases} 1, & \text{if } m^{q_i}_{j_{\min}} \leq m_j \leq m^{q_i}_{j_{\max}} \\ 0, & \text{otherwise} \end{cases} \tag{1c}$$

To understand if a service should be loaded off, we consider both its current SLO fulfillment as well as predictions according to historical observations; for this, we infer the predicted SLO fulfillment (Line 3) using a Bayesian Network (BN). BNs are structural causal models encoded as Directed Acyclic Graph (DAG); nodes represent random variables (e.g., *cpu*) and an edge between two variables (e.g., $cpu \rightarrow energy$) indicates conditional dependency, i.e., *cpu* influences the states of *energy*. Given historical observations, BNs can answer how likely it is to observe a specific (i.e., SLO fulfilling) state at runtime [23,24]; hence, we call them SLO interpretation (SLO-I) models. For an SLO-I model m and service s, agents predict SLO fulfillment through $\texttt{INFER}(m, s.Q, s.C)$.

To ensure that predictions remain accurate regardless of variable drifts, increasing prediction errors trigger retraining. As more training data is collected (Line 4), the utilization of the metrics buffer, as shown in Eq. (2), indicates that the model becomes outdated, putting additional weight on retraining.

$$\texttt{FULL}(B) = \frac{\sum_{i=1}^{n} 1}{|B|} \tag{2}$$

Next, in Line 5, we calculate the evidence to retrain (e_r) as the sum of absolute prediction error and metric buffer utilization. If e_r surpasses the retraining rate (ρ), the metrics buffer is sent to the platoon leader to update the SLO-I model; this is further elaborated in Sect. 3.2. Notice that both the maximum buffer size

Algorithm 1. Continuous SLO Interpretation

```
Require: $D$, $B$, $W_\phi$; $s$, $m_{s,t}$, $\rho$, $\omega$, $\gamma$ (global)
 1: $\phi_s \leftarrow \phi(s.Q)$
 2: $W_\phi \leftarrow W_\phi \cup \phi$
 3: $p_\phi \leftarrow \texttt{INFER}(m_{s,t}, s.Q, s.C)$
 4: $B \leftarrow B \cup D$
 5: $e_r \leftarrow \mathbf{abs}(W_\phi - p_\phi) + \texttt{FULL}(B)$
 6: if $e_r > \rho$ then
 7:     $m_{s,t} \leftarrow \texttt{RETRAIN}(B); B \leftarrow \emptyset$
 8: end if
 9: $e_o \leftarrow \mathbf{abs}(W_\phi - p_\phi)) + (1 - W_\phi)$
10: if $e_o > \omega$ then
11:     $v' \leftarrow \texttt{FIND\_OFFLOAD}(s, v)$
12:     if $v' \neq \emptyset$ then $\texttt{OFFLOAD}(s, v')$
13: end if
```

($|B|$) as well as ρ can be customized; for instance, $\rho = 1.0$ would be exceeded if FULL(B) $= 0.8$ and the prediction is off by 0.3.

Model retraining assures that offloading decisions are taken based on accurate assumptions; to that extent, the evidence to load off (e_o) is computed (Line 9) as the sum of absolute SLO violation and prediction error. When e_o surpasses a custom rate ω, and only in this case, does the agent look for a suitable host within the vehicle platoon (Line 11); given that there is one, the service will then be offloaded there; this will be explained in more detail in Sect. 3.3.

3.2 Collaborative Training

Retraining of SLO-I models is carried out by the platoon leader, i.e., a distinguished member elected; however, training data is provided by all platoon members. For instance, recall Fig. 2, where s_2 and s_5 are two CV service instances executed on different hosts. Each service collects evidence to retrain (e_r) independently of other instances; once its $e_r > \omega$, the service requests a model update from the platoon leader, providing its local training buffer. Technically, our architecture allows platoon members to update SLO-I models locally; however, limiting the training to the leader improves model consistency over the platoon, plus it isolates the training overhead. Also, to avoid a platoon leader becoming a single point of failure, new leaders can be reelected at any point; for the context of this work, we exclude leader election strategies from the analysis.

Each combination of service and device type is encoded in a unique SLO-I model. Therefore, as soon as the platoon leader (v_1) receives a metric buffer (B_{s,v_2}) from a member (v_2), it first checks v_2's type of processing device ($v_2.t$), e.g., Jetson Orin NX. Next, the leader updates its local SLO-I model ($m_{s,t}$) for service s and device type $v_2.t$; in our example, this means updating the SLO-I model of service $s = CV$ executed on device type $t = NX$. Finally, a new model version $m' = \texttt{PARL}(m, B)$ is created by updating the BN parameters according to recent observations ($B_{s,v}$). Retraining through PARL is limited to updating the

Algorithm 2. Evaluating Alternative Host (`FIND_OFFLOAD`)

```
Require: $s$, $v$; $P$, $A$, $M$ (global)
Ensure: $v'$ {Optimal vehicle for offloading $s$ from $v$}
 1: if $|P| = 1$ then return $\emptyset$
 2: $S_v \leftarrow \{s_a \mid (s_a, v_a) \in A \mid v_a = v\}$
 3: $S'_v \leftarrow S_v \setminus \{s\}$; $\Gamma \leftarrow \emptyset$
 4: $\phi_S \leftarrow \texttt{INFER}(M[S_v], S_v.Q, \texttt{CONV\_HW}(S_v, v.t))$
 5: $\phi_{S'} \leftarrow \texttt{INFER}(M[S'_v], S'_v.Q, \texttt{CONV\_HW}(S'_v, v.t))$
 6: for each $w$ in $P \setminus \{v\}$ do
 7:     $\Sigma_w \leftarrow \{s_a \mid (s_a, v_a) \in A \mid v_a = w\}$
 8:     $\Sigma'_w \leftarrow \Sigma_w \cup \{s\}$
 9:     $\phi_\Sigma \leftarrow \texttt{INFER}(M[\Sigma_w], \Sigma_w.Q, \texttt{CONV\_HW}(\Sigma_w, w.t))$
10:     $\phi_{\Sigma'} \leftarrow \texttt{INFER}(M[\Sigma'_w], \Sigma'_w.Q, \texttt{CONV\_HW}(\Sigma'_w, w.t))$
11:     $\gamma \leftarrow (\phi_{S'} + \phi_{\Sigma'}) - (\phi_S + \phi_\Sigma)$
12:     $\Gamma \leftarrow \Gamma \cup (\gamma, w)$
13: end for
14: $\gamma, v' \leftarrow \{(\gamma, w) \in \Gamma, \mathbf{max}(\gamma)\}$
15: return $v'$ if $\gamma > 0$ else $\emptyset$
```

conditional probabilities of BN variables; the structure (i.e., variable relations) is left untouched and only supplied through expert knowledge.

After retraining, the updated model (m') is shared within the platoon. For this, the platoon leader broadcasts $m'_{s,t}$ to all members in $\{v \in P \mid v.t = v_2.t\}$, i.e., to all platoon members with the matching device type. Vehicles that received an updated model now substitute the SLO-I models of locally running services. For Fig. 2, this would mean that s_2 gets updated, but s_5 not, since v_1 has a device type $v_2.t \neq v_1.t$. Thus, all instances of service s at vehicles with type $v.t = v_2$ interpret their SLO fulfillment according to the new model version.

3.3 Service Offloading

Once a service collected sufficient evidence to load off (e_o), like s_2 in Figs. 1 and 2, the service looks for the best alternative host, which means comparing for each of the other platoon members if global SLO fulfillment would be improved by offloading there. Formally, this is described in Algorithm 2, which uses the list of platoon members (P), the assignments (A) of which vehicle currently executes which service, and the shared collection (M) of all SLO-I models. In case the platoon does not contain other vehicles (Line 1), the search stops immediately; otherwise, the service predicts (1) the combined SLO fulfillment (ϕ_S) for all services (S_v) executed at vehicle v (Line 4), and (2) how offloading s would change local SLO fulfillment ($\phi_{S'}$) (Line 5). For this, we first estimate the combined hardware demand (`CONV_HW`) that would emerge from co-locating the services on a target device and then estimate per service if the increased hardware load has an impact on its SLO fulfillment.

Before continuing Algorithm 2, we briefly explain `CONV_HW`(S, t), which predicts the hardware utilization that would result from executing all $s \in S$ at a device of type t. For each service $s \in S$, we use the respective model $m_{s,t} \in M$ to infer its expected hardware utilization; in our case, we consider the hardware variables $hw = \{cpu, gpu, memory\}$, but the list can be extended arbitrarily with other monitor variables included in the SLO-I model. This returns a probability distribution (e.g., p_{cpu}) for each variable $\in hw$; afterward, the combined hardware load is calculated as the convolution of the individual loads. Formally, the convolution of two or more random variables (X, Y) with probability density functions $f_X(x)$ and $f_Y(y)$, i.e., the probabilities for each hw variable, is the sum $(Z = X + Y)$ of their individual distributions [2], as shown in Eq. (3).

$$f_Z(z) = (f_X * f_Y)(z) = \int_{-\infty}^{\infty} f_X(t) f_Y(z - t)\, dt \tag{3}$$

Thus, we obtain the combined hardware utilization, which is supplied as a constraint to `INFER`; this allows estimating how the respective hardware load would impact SLO fulfillment (ϕ_S and $\phi_{S'}$). Alternative approaches to estimating combined load and resulting SLO fulfillment might need to empirically test the service deployment, which is infeasible when decisions must be made quickly.

In the next step, we estimate for each of the other platoon members (w) the SLO fulfillment (ϕ_Σ) of its local services (Σ_w) and how this would be affected $(\phi_{\Sigma'})$ if we would offload s there. This follows the same pattern applied for the source vehicle v: we use the list of services executed at w (Line 7) and their respective SLO-I models to estimate their SLO fulfillment according to the combined hardware load (Lines 9 & 10). The last step is calculating the offloading gain (γ) for each platoon member (w), i.e., whether global SLO fulfillment would be improved by offloading s to w, and then return the best possible vehicle. For this, it first calculates γ (Line 11), which is appended to the collection Γ. In the final step, it selected the best alternative host among the platoon members (Line 14); however, if not even the best host would improve overall SLO fulfillment, it prefers to keep the current host (Line 15). The outcome is returned to Algorithm 1, which offloads the service accordingly.

4 Evaluation

Here, we evaluate our methodology for a set of heterogeneous perception services and a composable vehicle platoon. Specifically, we implement a prototype of our framework that addresses the illustrated scenario; afterward, we document the experimental setup, including service implementations and applied processing hardware, then present the experimental results, and critically discuss them.

4.1 Implementation

To implement our methodology, we provide a Python-based prototype[2] that follows a clear modular structure for services, their SLOs, and device types.

Hence, the framework can be extended with new services as long as they are supported by the underlying edge device. Once the framework is installed[3], services can be started or stopped remotely through HTTP; for running the experiments, we send the respective instructions to different platoon members using Postman flows. To isolate resource consumption, services are executed in individual Python threads. During that time, each service observes its SLO fulfillment as part of its service wrapper (i.e., Algorithm 1); in the present state, this is done every 500ms, though it can be customized for service types or instances. To avoid interfering with regular service execution, model training and evaluation of alternative service hosts run detached from the main service thread.[4]

Vehicles communicate exclusively over HTTP; the respective connection is established either through a local access point managed by the platoon leader, or through IBSS, i.e., a peer-to-peer network. Training and updating of SLO-I models, or rather their underlying BNs, uses pgmpy [1], a Python library for Bayesian Network Learning (BNL). In pgmpy, BNs can be encoded in XML, which each had a size of roughly 10kB in our evaluation; hence, a feasible size to be transmitted and shared within the platoon.

4.2 Experimental Setup

To evaluate our prototype in a realistic environment, we implement the scenario illustrated in Sect. 2.1, i.e., perception services are offloaded within a vehicle platoon according to their local SLO fulfillment. We provide three perception services that can be executed on edge devices; Table 1 provides essential information on these services: *CV* uses Yolov8 to detect objects in a video stream, *LI* processes point clouds from a Lidar sensor to map the environment, and *QR* uses OpenCV to detects QR codes in a video. Each service has specific tuning parameters, such as the resolution (*pixel*) and *fps* for *CV* and *QR*; *LI* accepts an additional parameter *mode* to define the point cloud radius.

According to our expert knowledge, each service's expected QoS level is specified through a list of SLOs; through heuristic trial and error, the following ones proved useful: we constrain the processing **time** ≤ 1000 / fps, i.e., frames must be processed faster than they come in; the maximum **energy** consumption can be adjusted for individual devices: we put a limit of ≤ 15 W for regular platoon members and ≤ 25 W for the platoon leader. Notice, that this considers the vehicle-wide energy consumption over all executed services. According to the video resolution (*pixel*) provided to *CV*, the service uses the respective Yolov8

[2] The framework prototype is available at GitHub, accessed on July 14th 2024.

[3] Postman is a common tool for sending HTTP requests; Postman flows is a UI extension that allows to specify sequences of requests, e.g., start/stop services.

[4] Instructions are provided in the following README, accessed on July 14th 2024.

Table 1. List of all predefined services that were added to the framework

ID	Service Description	CUDA	Parameters	SLOs
CV	Object Detection with Yolov8 [26]	Yes	*pixel*, *fps*	**time**, **energy**, **rate**
LI	Lidar Point Cloud Processing [6]	Yes	*mode*, *fps*	**time**, **energy**
QR	Detect QR Code w/ OpenCV [21]	No	*pixel*, *fps*	**time**, **energy**

Table 2. List of all edge devices that were involved in the evaluation

Full Device Name	ID	Price[a]	CPU	RAM	GPU	CUDA
Jetson Orin NX (3)	*NX*	450 €	ARM Cortex 8C	8 GB	Volta 1k	11.4
Jetson Orin AGX	*AGX*	800 €	ARM Cortex 12C	64 GB	Volta 2k	12.2

[a]Prices adopted from sparkfun, accessed Jul 14th 2024

model size (i.e., v8n, v8s, v8m); however, this affects the number of objects that are detected, which is ensured through the **rate** SLO.

The presented framework is evaluated on two different instances of Nvidia Jetson boards, namely Jetson Orin *NX* and Orin *AGX*, which are described in more detail in Table 2: the *AGX* is superior in terms of memory and GPU and has a slightly better CPU. While the specific Nvidia CUDA version has minor importance, CUDA itself is crucial to accelerate the *CV* and *LI* services. Each Jetson *NX* is embedded in a Rosmaster R2[5] car – a battery-powered multi-sensory vehicle used for development. To ensure a stable evaluation environment, the service processed either prerecorded videos (*CV* & *QR*) or binary-encoded point clouds (*LI*); Fig. 3 shows a demo output for each service.

4.3 Results

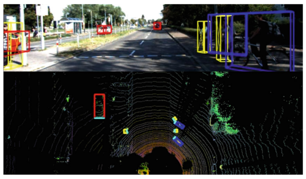
We evaluate the prototype by observing: (1) what is the overhead of continuously interpreting services, and what limitations arise from the platoon size; (2) if the SLO-aware retraining ensure prediction accuracy regardless of unexpected runtime behavior; and (3) if the framework fulfills high-level SLOs within the platoon by offloading computations. We assess these aspects using two base cases and one advanced scenario, all of which involve real workloads and devices:

Scenario 1A. An individual vehicle (i.e., *NX* or *AGX*) executes the *QR* service; every 25 s, we add a vehicle to its platoon, up to a maximum size of 4 vehicles. Given this, we track the time to execute the service wrapper, i.e., how long it takes to retrain the SLO-I model and evaluate alternative hosts for *QR*.

Figure 4 visualizes the times required to train the SLO-I model or evaluate alternative hosts for offloading; both processes are executed as part of the service wrapper. The wrapper runs every 500 ms for a total of 100 s, hence, the plot

[5] More information about the Rosmaster R2 here, accessed Jul 14th 2024.

(a) CV (Yolov8)


(b) LIdar (SFA3D)

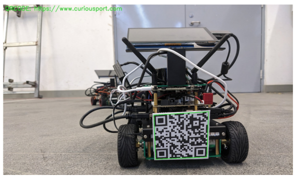
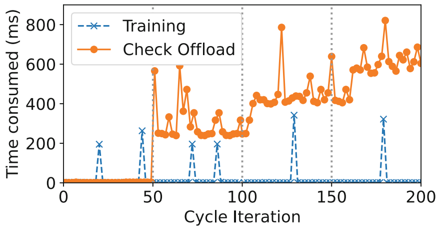
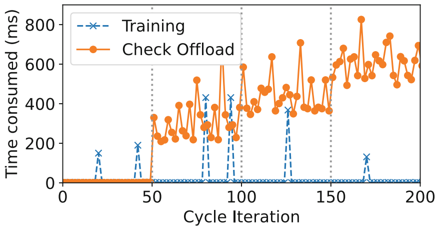

(c) QR (OpenCV)

Fig. 3. Demo output for each service according to the prerecorded input data



(a) Orin *NX*



(b) Orin *AGX*

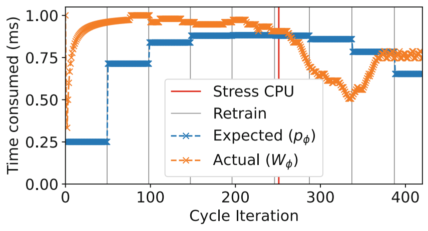
Fig. 4. Time required to train the SLO-I model and evaluate alternative hosts

contains 200 wrapper iterations. Vertical grey lines indicate when an additional device is introduced to the platoon, i.e., at 50, 100, and 150 iterations.

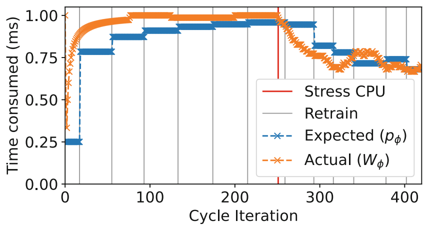
Given this, we conclude that the platoon size has a linear impact on the time required to evaluate alternative hosts; the exception is $|P| = 1$, when evaluating other vehicles for offloading is obsolete. For a platoon with $|P| \leq 3$, the entire service wrapper finished mostly in ≤ 500 ms; however, $|P| \geq 4$ starts exceeding 500 ms, which indicates that it would not be possible to interpret the SLO fulfillment every 500 ms. This could be overcome by either structuring the platoon into smaller subgroups or adjusting the evaluation interval.

Scenario 1B. An individual vehicle (i.e., AGX) runs *CV* locally; however, the respective SLO-I model was not yet fine-tuned and initial predictions are likely inaccurate. Additionally, variable drifts occur, which we simulate through stress-ng: after 125 s the CPU load of *AGX* is stressed 40%. We measure p_ϕ and W_ϕ, and compare our presented training strategy with a static service wrapper.

Figure 5 visualizes for both runs the predicted (p_ϕ) and actual SLO fulfillment (W_ϕ); vertical grey lines indicate when retraining happened, and the red line when the perturbation occurred. Not only does the left side perform fewer retraining, i.e., 8 instead of 12, but more importantly, the right side presents shorter training intervals when the SLO fulfillment is unstable, such as during the period between $x = [250, 350]$. Consequentially, the Mean Squared Error (MSE) was 0.07 on the left and 0.01 on the right side; given that, we conclude that SLO-dependent retaining helped to increase the prediction accuracy for initially inaccurate models or at runtime when perturbations occur.

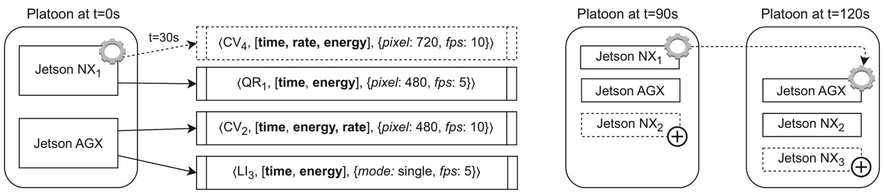


(a) Without SLO-dependent retraining



(b) With SLO-dependent retraining

Fig. 5. Improved prediction accuracy through SLO-dependent retraining



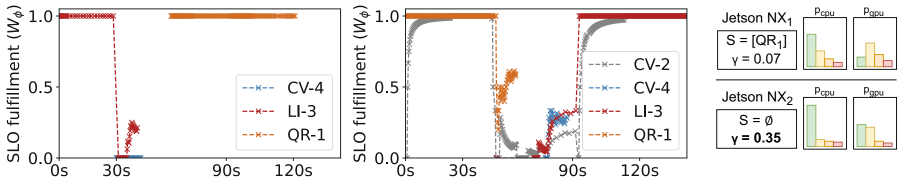
Platoon at t=0s
Jetson NX$_1$
Jetson AGX
t=30s
⟨CV$_4$, [time, rate, energy], {pixel: 720, fps: 10}⟩
⟨QR$_1$, [time, energy], {pixel: 480, fps: 5}⟩
⟨CV$_2$, [time, energy, rate], {pixel: 480, fps: 10}⟩
⟨LI$_3$, [time, energy], {mode: single, fps: 5}⟩
Platoon at t=90s
Jetson NX$_1$
Jetson AGX
Jetson NX$_2$
Platoon at t=120s
Jetson AGX
Jetson NX$_2$
Jetson NX$_3$

Fig. 6. Sequential description of Scenario 2: starting services and adjusting the platoon

Scenario 2. Figure 6 provides a sequential description of this scenario: at time $t = 0s$ the platoon $P = \{NX_1, AGX\}$ starts 3 services (i.e., QR_1, CV_2, LI_3); at $t = 30$ s NX_1 starts CV_4; at $t = 90$ s NX_2 joins the platoon, and at $t = 120$ s NX_3 joins, NX_1 leaves the platoon, and leadership is transferred to AGX.

Figure 7 visualizes the SLO fulfillment of all services executed at NX_1 and AGX; at first, all three services (i.e., QR_1, CV_2, LI_3) achieve maximum SLO fulfillment, i.e., $W_\phi = 1.0$. However, as soon as CV_4 is started at $t = 30$ s, NX_1 fails to ensure the SLOs for both LI_3 and CV_4. Due to that, NX_1 decides to load off both services to AGX, which in turn, causes AGX to fail most of its services' SLOs. This changes at $t = 55$ s, when AGX decides to move one of its services (i.e., $QR\text{-}1$) to NX_1, which slightly recovers the SLO fulfillment of the remaining three services. Next, at $t = 90$ s, NX_2 joins the platoon, which encourages AGX to offload another service (i.e., CV_4) to NX_2. Here, Fig. 7c shows the decision-making of AGX: since NX_1 already executes QR_1, it estimates how adding CV_4 would have a negative impact on QR_1 due to predicted resource shortage; hence, it chooses NX_2, which promises global SLO improvement of $\gamma = 0.35$.

Given this, we conclude that services can react in ≤ 10 s to local SLO violations, which appears practical for real-time systems. This highlights the impact of co-locating too many services at one edge device and how this can be resolved by adding new vehicles to the platoon. Furthermore, changing the platoon leader at $t = 120$ showed no negative impact on the remaining vehicles – its ongoing computations were shifted to an idle vehicle (i.e., NX_4) that just had joined.



(a) Orin NX_1 (b) Orin AGX (c) CV_4 at $t = 90s$

Fig. 7. SLO fulfillment and decision making for constrained services in the platoon

5 Conclusion and Future Work

This paper introduced a novel V2V offloading mechanism that ensures high-level requirements during runtime. By leveraging probabilistic models, individual services can estimate the resource demand over multiple services and the consequential SLO fulfillment at alternative hosts. We evaluated the proposed framework in a physical setup, in which platoon members feature heterogeneous processing devices. Noteworthy, we showed how the framework could handle an increasing number of platoon members and a series of perception services; hence, it improves platoon-wide SLO fulfillment through decentralized decision-making. While our work showed promising results, there remain limitations and areas of improvement: First, although baselines are scarce, the work must be contrasted with comparable approaches to provide further insights. While we ruled network latency negligible in our case, future work could also include this for more detailed analyses. Furthermore, our implementation executes services in Python threads; we plan to implement a more effective and elegant solution, containerizing each service instance. Another interesting direction would be to explore more complex architectures in which a single platoon has multiple swarms or when multiple platoons need to coordinate with each other.

References

1. Ankan, A., Textor, J.: pgmpy: A Python Toolkit for Bayesian Networks (2023)
2. Bacchus, F.I.: Representing and Reasoning with Probabilistic Knowledge, Artificial Intelligence. MIT Press, Cambridge (1989)
3. Chen, C., et al.: Delay-optimized V2V-based computation offloading in urban vehicular edge computing and networks. IEEE Access **8**, 18863–18873 (2020)
4. Dong, S., Xia, Y., Kamruzzaman, J.: Quantum particle swarm optimization for task offloading in mobile edge computing. IEEE TII (2023)
5. Du, H., Leng, S., Zhang, K., Zhou, L.: Cooperative sensing and task offloading for autonomous platoons. In: IEEE GLOBECOM 2020 (2020)
6. Dzung, N.M.: maudzung/SFA3D (2020). https://github.com/maudzung/SFA3D
7. Fan, W., et al.: Joint task offloading and resource allocation for vehicular edge computing based on V2I and V2V modes. IEEE Trans. Intell. Transp. Syst. (2023)
8. Fan, X., Cui, T., Cao, C., Chen, Q., Kwak, K.S.: Minimum-cost offloading for collaborative task execution of MEC-assisted platooning. Sensors (2019)

9. Gao, Y., Liu, L., Zheng, X., Zhang, C., Ma, H.: Federated sensing: edge-cloud elastic collaborative learning for intelligent sensing. IEEE Internet Things (2021)
10. Guo, H., Liu, J., Lv, J.: Toward intelligent task offloading at the edge. IEEE Netw. **34**(2), 128–134 (2020). https://doi.org/10.1109/MNET.001.1900200
11. Guo, H., Rui, L.l., Gao, Z.P.: V2V task offloading algorithm with LSTM-based spatiotemporal trajectory prediction model in SVCNs. IEEE TVT (2022)
12. Hazra, A., Donta, P.K., Amgoth, T., Dustdar, S.: Cooperative transmission scheduling and computation offloading with collaboration of fog and cloud for industrial IoT applications. IEEE Internet Things J. (2023)
13. Hu, Y., Cui, T., Huang, X., Chen, Q.: Task offloading based on lyapunov optimization for MEC-assisted platooning. In: 2019 11th International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1–5 (2019)
14. Kuutti, S., Bowden, R., Jin, Y., Barber, P., Fallah, S.: A survey of deep learning applications to autonomous vehicle control. IEEE TITS (2020)
15. Le Mero, L., Yi, D., Dianati, M., Mouzakitis, A.: A survey on imitation learning techniques for end-to-end autonomous vehicles. IEEE Trans. Intell. Transp. Syst. **23**(9), 14128–14147 (2022)
16. Lu, L., Li, X., Sun, J., Yang, Z.: Cooperative computation offloading and resource management for vehicle platoon: a deep reinforcement learning approach. In: IEEE International Conference on High Performance Computing and Communications (2022)
17. Ma, X., Su, Z., Xu, Q., Ying, B.: Edge computing and UAV swarm cooperative task offloading in vehicular networks. In: 2022 International Wireless Communications and Mobile Computing (IWCMC), pp. 955–960 (2022)
18. Ma, Y., Wang, Z., Yang, H., Yang, L.: Artificial intelligence applications in the development of autonomous vehicles. IEEE J. Automatica Sinica (2020)
19. Morichetta, A., Spring, N., Raith, P., Dustdar, S.: Intent-based management for the distributed computing continuum. In: 2023 IEEE International Conference on Service-Oriented System Engineering (SOSE), pp. 239–249. IEEE (2023)
20. Mousa, M.H., Hussein, M.K.: Efficient UAV-based mobile edge computing using differential evolution and ant colony optimization. PeerJ Comput. Sci. (2022)
21. OpenCV: OpenCV at 4.9.0 (2024). https://github.com/opencv/opencv/tree/4.9.0
22. Qiao, G., Leng, S., Zhang, K., He, Y.: Collaborative task offloading in vehicular edge multi-access networks. IEEE Commun. Mag. **56**(8), 48–54 (2018)
23. Sedlak, B., Pujol, V.C., Donta, P.K., Dustdar, S.: Equilibrium in the computing continuum through active inference. Future Gener. Comput. Syst. (2024)
24. Sedlak, B., Pujol, V.C., Donta, P.K., Dustdar, S.: Diffusing high-level SLO in microservice pipelines. In: IEEE SOSE (2024)
25. Tasoulas, V., Haugerud, H., Begnum, K.M.: Bayllocator: A Proactive System to Predict Server Utilization and Dynamically Allocate Memory Resources Using Bayesian Networks and Ballooning (2012)
26. Varghese, R., M., S.: YOLOv8: a novel object detection algorithm with enhanced performance and robustness. In: ADICS (2024)
27. Wu, Y., et al.: Intelligent resource allocation scheme for cloud-edge-end framework aided multi-source data stream. EURASIP J. Adv. Sig. Process. **2023** (2023)
28. Xu, B., et al.: Optimization of cooperative offloading model with cost consideration in mobile edge computing. Soft. Comput. **27**(12), 8233–8243 (2023)
29. Zhang, Z., Jiang, J., Xu, H., Zhang, W.A.: Distributed dynamic task allocation for unmanned aerial vehicle swarm systems: a networked evolutionary game-theoretic approach. Chin. J. Aeronaut. (2023)