

CNN-assisted Road Sign Inspection on the Computing Continuum

Narges Mehran and Radu Prodan

Institute of Information Technology, Alpen-Adria-Universität Klagenfurt, Austria

Email: {name}.{surname}@aau.at

Abstract—Processing rapidly growing data encompasses complex workflows that utilize the Cloud for high-performance computing and the Fog and Edge devices for low-latency communication. For example, autonomous driving applications require inspection, recognition, and classification of road signs for safety inspection assessments, especially on crowded roads. Such applications are among the famous research and industrial exploration topics in computer vision and machine learning. In this work, we design a road sign inspection workflow consisting of 1) encoding and framing tasks of video streams captured by camera sensors embedded in the vehicles, and 2) convolutional neural network (CNN) training and inference models for accurate visual object recognition. We explore a matching theoretic algorithm named CODA [1] to place the workflow on the computing continuum, targeting the workflow processing time, data transfer intensity, and energy consumption as objectives. Evaluation results on a real computing continuum testbed federated among four Cloud, Fog, and Edge providers reveal that CODA achieves 50 %-60 % lower completion time, 33 %-59 % lower CO₂ emissions, and 19 %-45 % lower data transfer intensity compared to two state-of-the-art methods.

Index Terms—Computing continuum, machine learning, Cloud, Fog, Edge, placement.

I. INTRODUCTION

International data corporation predicts that 21.5 billion connected Internet of Things (IoT) devices will generate 55 % of all data by 2025 [2]. This vast amount of data often encompasses high computational workloads requiring low communication and processing latency using machine learning (ML) [3] technologies. Traditional workflow scheduling methods use the Cloud, Fog, and Edge resources in isolation, leading to a complex deployment process [4]. Furthermore, they ignore the ML training along with the video encoding and framing requirements and explore only specific inference tasks [4] processed at the Fog and Edge. Therefore, challenges in automating ML processing in the computing continuum still exist, including allocation and orchestration.

As autonomous driving [5] is slowly becoming a reality, automating the inspection and recognition of road signs, captured using camera sensors installed on the dashboard of vehicles and driver assistance systems, is imperative. We, therefore, explore in this paper a road sign inspection application that receives a video stream from a camera sensor and recognizes the road sign in the appropriate video frame. We design a workflow that pre-processes the sensed data using two high-quality video encoding and framing tasks. Afterward, we apply a convolutional neural network (CNN)

model for the recognition and classification of road signs customized to the computational capability of the computing continuum resources: high training accuracy in the Cloud and low training accuracy at the Edge. We implemented a 9-layer CNN, modeled according to a large dataset consisting of 43 German traffic road signs [6].

We explore the placement of the road sign inspection workflow on the computing continuum using the CODA algorithm [1] based on matching theoretic principles involving two sets of players negotiating different requirements:

- *Road sign inspection tasks* rank the continuum devices based on their data processing times;
- *Cloud, Fog, and Edge devices* rank the road sign inspection tasks based on their data transfer intensity.

CODA allocates the tasks to devices based on mutual preferences, aiming to maximize the aggregate profit of the CNN-assisted workflow and the resource providers [1]. A comparative evaluation indicates that CODA lowers the completion time by 50 %-60 %, the CO₂ emissions by 33 %-59 %, and the data transfer intensity by 19 %-45 % compared to two state-of-the-art methods.

The paper has seven sections. We survey the related work in Section II. Section III describes the design and implementation of the CNN-assisted road sign inspection workflow. Afterward, Section IV elaborates on the CODA architecture for workflow placement on the computing continuum. Section V describes the experimental design of a real-world evaluation, followed by the results in Section VI. Section VII concludes the paper.

II. RELATED WORK

This section reviews the state-of-the-art for placing workflow applications on Cloud, Fog, and Edge infrastructures.

1) *Machine learning on the computing continuum*: Lujic et al. [4] designed a scheduling method named SEA-LEAP that uses a pre-trained CNN for on-demand object detection in a traffic and pedestrian safety application with optimized completion time and Edge device locality to data producers. Nikouei et al. [7] explored CNN models on the Edge for object detection and tracking in delay-sensitive and mission-critical applications, like smart surveillance. However, these methods ignore video encoding and framing task placements.

2) *Autonomous driving application on the Fog*: Yu et al. [8] investigated an integer linear programming method that minimizes the deployment cost of an autonomous driving application on the Fog based on simulation analysis. However,

this method ignores the study of its components on a real testbed, including the ML-based one.

3) *Data processing time reduction*: Menouer [9] presented a container placement model based on the order of preference, similar to the ideal decision-making algorithm (TOPSIS), optimizing resource utilization and energy consumption. Mahmud et al. [10] proposed a Cloud-centric method that schedules tasks onto appropriate instances based on their minimum processing time. Although these works employ multi-criteria scheduling methods, they neglect the data transfer intensity and latency from the data producers to the Cloud in an extended Fog and Edge environment.

4) *Data transfer reduction*: Najafabadi Samani [11] proposed a multilayer partitioning method to minimize the resource wastage of Fog infrastructure that selects devices in resource partitions closest to the end-user. However, this work focuses on latency-sensitive workflows in Fog and Edge, isolated from the Cloud.

5) *Energy consumption*: Pooranian et al. [12] designed a penalty-aware heuristic algorithm to preserve load balancing in Fog by assigning negative scores that avoid data centers with exponential energy consumption. You et al. [13] proposed an energy-efficient asynchronous mobile Edge computation offloading method targeting a joint data partitioning and time division policy. These works fail to address the heterogeneous data arrival rate in processing workflows and neglect the load produced by large data exchanges and the device utilization during their execution in the computing continuum.

III. ROAD SIGN INSPECTION

We selected a road sign inspection workflow following safety concerns explained in previous related work [14], [1]. Afterward, we extended it with asynchronous communication between tasks facilitated by message queues [15], following an Alibaba analysis revealing that 23% of the communication between microservices use indirect data exchange [16]. Such data queues temporarily store the video stream or frames traversing between the tasks.

A. Workflow design

The road sign inspection workflow receives a raw video, encodes it in high resolution, divides it into frames, and detects the sign inside the video based on the applied CNN model. Figure 1 represents the workflow, consisting of eight tasks. We summarize the workflow requirements per task in Table I, comprising the CPU (in MI), memory, and storage (in GB).

- *Encoding* a raw video using FFmpeg with H.264 codec;
- *Framing* video source to still images to using OpenCV;
- *Training* the CNN model for road sign classification with different low and high accuracy;
- *Inference* of the trained model with various accuracy;
- *Packaging* the detected sign in a required format (i.e., recognized sign image);
- *Dataset storage* of road sign records for inventory purposes;
- *Consumer* of warnings related to the roadside traffic signs.

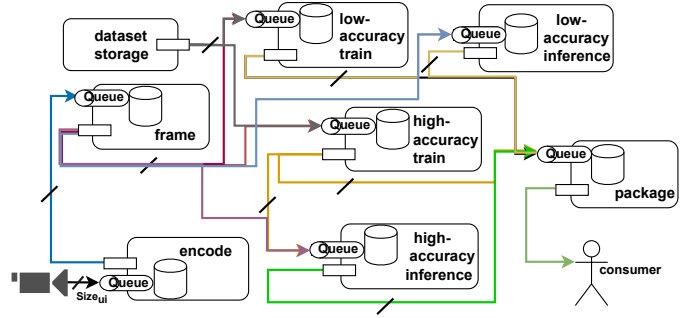


Fig. 1: Road sign inspection workflow.

TABLE I: Workflow resource requirements per task.

	CPU [MI] · 10 ³	Memory [GB]	Storage [GB]
<i>encoding</i>	300-400	0.5-1	0.05-0.1
<i>framing</i>	100-200	0.1-0.3	0.1-0.5
<i>low-accuracy train</i>	400-500	0.5-2	0.5-1
<i>low-accuracy inf.</i>	100-200	0.2-0.5	0.5-1
<i>high-accuracy train</i>	400-500	0.5-2	0.5-1
<i>high-accuracy inf.</i>	100-200	0.3-0.5	0.5-1
<i>packaging</i>	100-200	0.1-0.3	0.1-0.5

B. CNN model

The CNN model uses 50 000 still frames for classification of 43 signs with an accuracy between 70%-90% (see Figure 2). The input to the first convolutional layer consists of still frames of size $32 \times 32 \times 1$. The first layer has a depth of 6. Since the filter size is (5,5) and the stride (1,1), the height and width of the images are $32 - 5 + 1 = 28$. Afterward, the max polling layer downsizes the image size and selects the max value pixel between any two adjacent ones. Hence, the second convolutional layer receives the images as input with the sizes of $14 \times 14 \times 6$ from its previous max polling layer. The second convolutional layer has depth of 16 while the height and width of output image are $(14 - 5 + 1) = 10$, resulting in the size of $10 \times 10 \times 16$. Another max pooling layer applied to the images reduces the size to $5 \times 5 \times 16$. Afterward, the CNN model flattens the image size to $5 \times 5 \times 16 = 400 \times 1$. This model then applies two fully connected layers of sizes 120 and 84. The dropout layer aims to reduce the overfitted model. Finally, the CNN returns a fully connected layer of size 43, equal to the number of road sign classes. The *softmax* activation function returns the probability of each signing class.

C. Workflow implementation

We implemented the workflow orchestration mechanism using *shell* and *Python* programming languages that exchange data between tasks using asynchronous queues. We deployed the workflow tasks in Docker containers on the computing continuum devices. We published the workflow source code in the GitHub repository¹.

1) *Docker images*: of the workflow tasks are available in the public Docker Hub repository. Table II summarizes

¹<https://github.com/SiNa88/Road-sign-inspection>

TABLE II: Docker images of road sign inspection tasks.

Task	x86 image	Arm image
Encoding	sina88/encoding:amd64	sina88/encoding:rpi4
Framing	sina88/framing:amd64	sina88/framing:rpi4
Training	sina88/training:amd64	sina88/lite-training:rpi4
Inference	sina88/inference:amd64	sina88/inference:rpi4

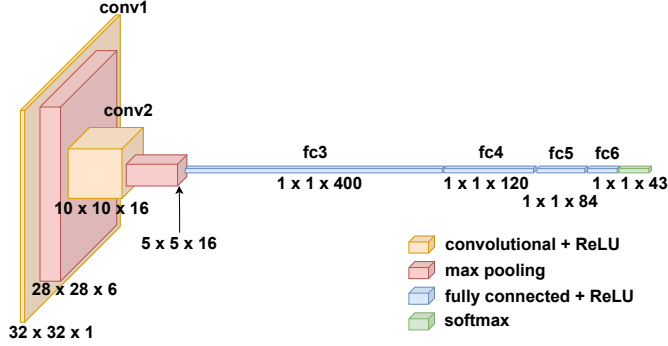


Fig. 2: CNN architecture.

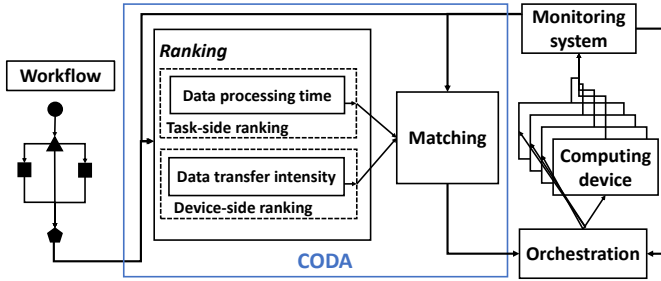


Fig. 3: CODA placement architecture.

the images created for two hardware architectures: x86 required by the Cloud servers and Arm supported by the Raspberry Pi devices at the Edge. We used the official images of Ubuntu 20.04, Debian buster-2020.11.17, and Python 3.9 to create the Docker images of different tasks. We extended the base images by installing packages such as Tensorflow v2.2.0 for linux_armv7l, FFmpeg v3.4.5, and OpenCV v4.5.0.

2) *Data queues*: implement asynchronous workflow tasks in which containerized tasks concurrently process the data to reduce the workflow completion time. We implemented asynchronous communication based on 1) Kubernetes KubeMQ [17] for the Edge devices, and 2) ZeroMQ for the Cloud instances.

IV. CODA PLACEMENT METHOD

Figure 3 illustrates the CODA placement architecture, consisting of four components.

1) *Ranking*: calculates the *device preference lists* for tasks ordered by their data processing time and the *task preference lists* for devices based on the data transferred to each task after resource allocation to avoid zero bandwidth surplus. For instance, the high-accuracy training task ranks AAU-medium as the preferred Fog instance. Since AAU-medium also has

this task in its preference list, a resource allocation match occurs. The AAU-large instance does not rank this task because of insufficient residual bandwidth after the allocation.

2) *CODA*: uses game theoretic principles that consider the tasks and device preference lists as *players* and applies a stable *two-sided matching algorithm* to place each task onto one device. The algorithm loops until it finds an appropriate device matching every task. Each device can be in the under-utilization, over-utilization, or full-utilization state. Firstly, a task matches its highest-ranked under-utilized device. If the device can host multiple tasks, the matching continues until it reaches its over-utilized state. Afterward, CODA removes the lowest-ranked matching of the device. If the device is in its full-utilization state, it follows the game principles to reach a stable matching and removes low-ranked tasks from its preference list, while the corresponding task removes the device from its list. At the end of the game, there is no blocking pair of a task and a device that prefers another matching to their current one. We implemented the CODA in Python 3.9 using the matching library published in [18]. We integrated our customized scheduler in the Kubernetes 1.21 orchestration tool using the Python client library 17.17 for Kubernetes [19]. The script required to run CODA is available in the GitHub code repository².

3) *Orchestration*: replaces the Kubernetes default scheduler with the CODA placement algorithm and deploys the tasks on the computing devices based on their earliest start time using the Docker images summarized in Table II. Afterward, it manages the workflow execution and the data transfer between tasks according to the workflow precedence relationships.

4) *Monitoring system*: observes the performance of the data processing over the devices and reports the information to the CODA orchestrator. We employed the Prometheus operator v0.45.0 to monitor the Kubernetes deployments, services, and devices [20]. We used the Prometheus queries for collecting the metrics, such as the utilization of a containerized task and its received data. Figure 4 shows the CPU utilization of the high-accuracy training task, which gradually increases after its deployment. In addition, the network ingress traffic increases as the high-accuracy training tasks receive the data from the dataset storage and communicate with the successor tasks in the workflow (see Figure 1).

V. EXPERIMENTAL DESIGN

A. Computing continuum testbed

We created a computing testbed [21] of heterogeneous virtual resource instances from four providers distributed across the Cloud, Fog, and Edge layers, displayed in Table III.

1) *Cloud layer*: consists of virtual machine instances provisioned on demand from two providers: (a) t2.xlarge from Amazon Web Services (AWS) with four virtual cores, 16 GB of memory and up to 10 Gbit/s network bandwidth; (b) n2-standard-4 from Google Cloud Platform (GCP)

²<https://github.com/DataCloud-project/ADA-PIPE/tree/main/ADA-match-scheduler>

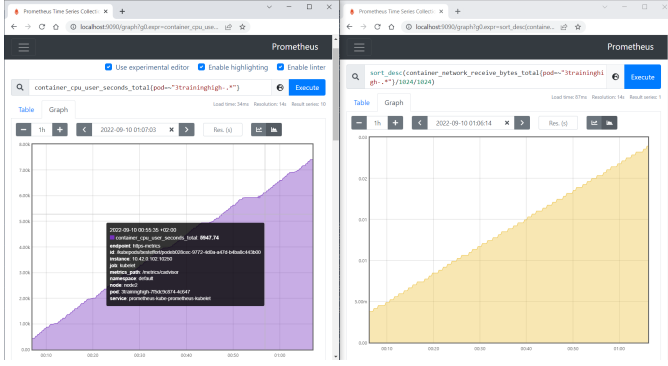


Fig. 4: CPU utilization and ingress traffic monitoring of the high-accuracy training task.

with four virtual cores, 16 GB of memory and up to 10 Gbit/s network bandwidth.

2) *Fog layer*: comprises two on-demand instances from the Exoscale provider, hosted in the data center of the A1 network operator in Vienna with a 10 Gbit/s network throughput: (a) large with four virtual cores and 8 GB of memory; (b) medium with two virtual cores and 4 GB of memory.

3) *Edge layer*: comprises the following devices interconnected by a managed layer-3 HP Aruba with 48 1 Gbit/s ports, 3.8 μ s latency and aggregate throughput of 104 Gbit/s: (a) NVIDIA Jetson Nano (NJN) (five devices) running Linux for Tegra (L4T) operating system; (b) Raspberry Pi (i.e., three RPi3 model B+ and 32 RPi4) running Raspberry Pi OS; (c) large virtual instance with a twelve-core processor and 32 GB of memory; (d) medium virtual instance with an eight-core processor and 16 GB of memory.

B. Experiments

We designed two experiments investigating the workflow processing and data transfer characteristics based on raw videos with multiple road signs [22]. We average the results over ten executions for statistical significance.

a) *Encoding bitrate experiments*: investigate the processing impact of encoding a raw video segment using five encoding bitrates of 200, 1500, 3000, 6500, and 20 000 kbit/s, corresponding to the five resolutions: 256×144 , 1024×576 , 1280×720 , 2560×1440 , and 3840×2160 pixels. We considered two CNN models with 70 % and 90 % training accuracy and different processing requirements.

b) *Frame size experiments*: compare the methods using frame sizes of 35, 300, 420, 1350, and 2560 kB, corresponding to five video frame resolutions.

C. Evaluation metrics

We evaluate the placement of the road sign inspection workflow on the computing testbed using three metrics.

a) *Completion time*: (in s) is the latest completion time among all its tasks preceding a consumer of the workflow [1].

b) *Data transfer intensity*: quantifies the overall occupancy of all network channels in the computing continuum. An intensity below one represents an under-utilized network, while an intensity above one indicates no more bandwidth and increasing queuing times.

c) *CO₂ emission*: (in g) during workflow execution account for the transmission and processing of all data [23]. We obtained the power consumption using direct measurements on the computing testbed [21] and other public Cloud power models (see Table III). We further assume that producing 1 kW h of energy emits 620 g of CO₂ [24].

D. Related work comparison

We selected two state-of-the-art methods for comparative evaluation tailored for application placement on the computing continuum: 1) *SEA-LEAP* [4] implements a heuristic algorithm to perform workflow and data movement over a placement control mechanism. 2) *Kubernetes container scheduling strategy (KCSS)* [9] schedules all tasks on Cloud or Fog data centers using the TOPSIS algorithm.

VI. EXPERIMENTAL RESULTS

Table IV summarizes the CODA, SEA-LEAP, and KCSS placements of the road sign inspection workflow on the infrastructure testbed described in Section V-A. While KCSS deploys the tasks onto clustered Cloud instances, SEA-LEAP maps each task separately and rarely selects the Edge devices. In contrast, CODA distributes the tasks onto the Fog and Edge devices, targeting the local large and medium instances for the CNN-based training and inference tasks.

1) *Encoding bitrate experiments*: evaluate the performance of CODA for various bitrate and resolution pairs.

a) *Completion time*: Figure 5a shows that CODA reduces the completion time by 51% compared to SEA-LEAP, primarily by selecting under-utilized Edge and Fog instances that provide lower data processing time and data transfer specifically for encoding and training tasks. KCSS schedules the tasks on Cloud instances without considering their higher transmission time and achieves an average of 59 % higher completion times. Unlike the related methods, CODA avoids devices with limited network bandwidth or processing capacity for CNN training tasks that require transferring large amounts of data for starting their high workload execution.

b) *Data transfer intensity*: Figure 5b shows that CODA induces 19 %, respectively 45 %, lower data transfer intensity than SEA-LEAP and KCSS. These related methods place the workflows on Edge or Fog instances close to the data producer or deploy all the tasks on the Cloud with expensive data transfers. In contrast, CODA considers the ingress bandwidth when placing the inter-dependent tasks generating lower data transfer. The encoding bitrates do not affect the data transfer in this experiment that considers a fixed data size.

c) *CO₂ emission*: Figure 5c shows that CODA reduces the CO₂ emissions by selecting the A1/Exoscale Fog provider or nearby Edge devices instead of AWS or GCP Cloud instances. All methods perform similarly for lower bitrates up

TABLE III: Experimental infrastructure testbed.

Instance	Cloud		Fog		Edge				
	AWS t2.xlarge	GCP n2-standard-4	A1/Exoscale large	A1/Exoscale medium	AAU large	AAU medium	NJN	RPi4	RPi3
CPU [cores]	4	4	4	2	12	8	4	4	4
Memory [GB]	16	16	8	4	32	16	4	4	1
Storage [GB]	8	8	10	10	32	32	16	16	16
Bandwidth [Mbit/s]	100	870	840	839	920	900	450	800	328
Latency [ms]	101	23	11.5	11.9	0.3	0.3	1	0.4	1
CO ₂ emission [mg/s]	40	30	29	21.4	23.1	19.2	1.05	0.7	0.6

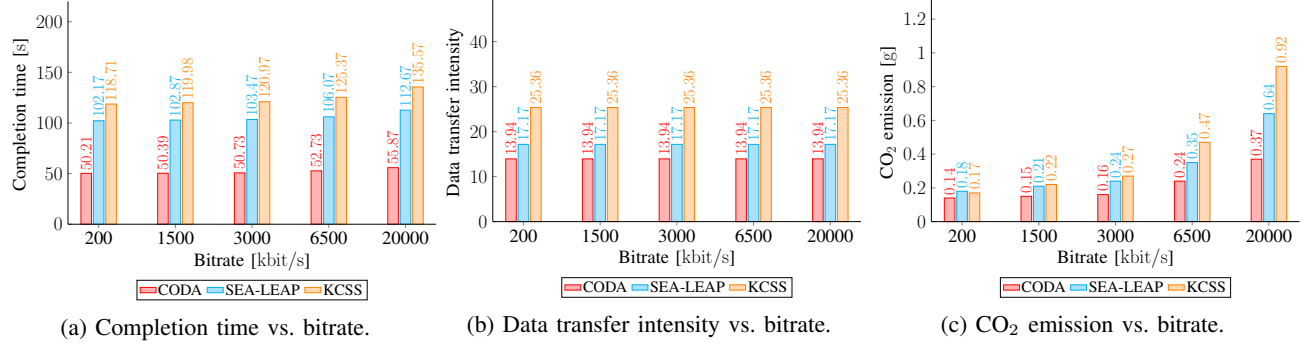


Fig. 5: Encoding bitrate experimental results for road signs inspection application.

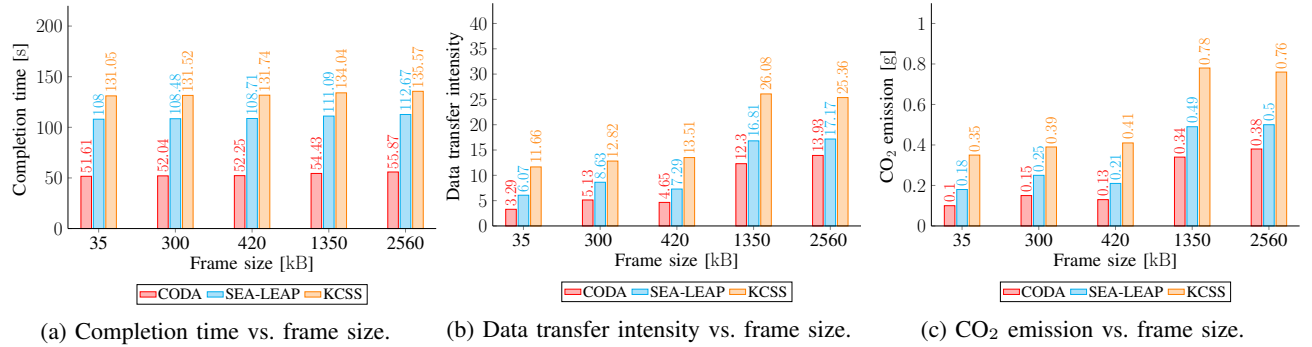


Fig. 6: Frame size experimental results for road signs inspection application.

TABLE IV: Placements of the road sign inspection tasks.

Task \ Method	CODA	SEA-LEAP	KCSS
Encoding	AAU medium	Exoscale large	GCP n2-standard-4
Framing	Exoscale large	Exoscale large	GCP n2-standard-4
High-accuracy training	AAU medium	Exoscale large	GCP n2-standard-4
High-accuracy inference	AAU medium	Exoscale medium	GCP n2-standard-4
Low-accuracy training	AAU large	AAU large	AWS t2.xlarge
Low-accuracy inference	AAU large	AAU medium	AWS t2.xlarge
Packaging	RPi4	AAU medium	AWS t2.xlarge

to 6500 kbit/s. CODA decreases the CO₂ by 50 % for a high bitrate of 20000 kbit/s, as the lower data processing times and data transfer, and the selection of Edge devices reduce

the overall energy consumption.

2) *Frame size experiments*: evaluate the performance of CODA for different frame sizes.

a) *Completion time*: Figure 6a shows that the frame size highly impacts the size of transmitted data, especially among encoding, framing, and training tasks (despite the fixed-size dataset). Furthermore, CODA reduced the completion time by 60 % compared to KCSS through improved resource utilization. While the latency-aware SEA-LEAP method allocates the data-intensive tasks to a Fog device close to the data producer, CODA balances the processing and transmission loads on a subset of under-utilized Edge devices, achieving 50 % lower completion times than SEA-LEAP.

b) *Data transfer intensity*: Figure 6b shows that a larger frame size generates a higher data transfer intensity. CODA transfers data among neighboring Fog and Edge instances, which lowers the data transfer over the network channels by 22 % compared to SEA-LEAP, and by 35 % compared to KCSS. Although the related methods consolidate the workflow

to a few devices, they do not consider the network distance to the data producers, bandwidth, and latency, leading to frequent and expensive data exchanges.

c) *CO₂ emission*: Figure 6c shows that CODA reduces the CO₂ emissions by 33 % and 59 % compared to SEA-LEAP and KCSS. KCSS selects AWS and GCP instances that consume more energy for transmitting data from producers to tasks. SEA-LEAP over-provisions certain Fog or Edge instances, especially for CNN training tasks, which considerably reduces the energy consumption compared to KCSS. CODA improves the utilization of low-powered single-board Edge devices along with Fog instances and lowers the CO₂ emissions, as they do not have to spend energy in idle mode.

VII. CONCLUSION AND FUTURE WORK

We designed a road sign inspection workflow consisting of encoding and framing tasks of video streams captured by camera sensors embedded in the vehicles, along with CNN training and inference models for object recognition. We applied the CODA matching-based method that considers the processing time and data transfer intensity to place its tasks on the computing continuum. We evaluated the workflow placements using two sets of experiments varying the data processing and transfer sizes in a real computing testbed with four federated providers across Cloud, Fog, and Edge. The results indicate that CODA achieved 50 %-60 % faster completion time and 33 %-59 % energy savings compared to two related methods. The gain results from the Fog and Edge federation by placing the high workload CNN training tasks to the most powerful Edge devices collocated with the inference tasks and the lighter packaging tasks on lower capacity devices. In the future, we plan to explore the benefit of executing CNN-assisted workflows in the computing continuum, focusing on lightweight models [7].

ACKNOWLEDGEMENT

This work received funding from:

- *European Union's* Horizon 2020 research and innovation program, grant agreement 101016835 (DataCloud);
- *Austrian Research Promotion Agency (FFG)*, grant agreement 888098 (Kärntner Fog) and grant agreement FO999897846 (GAIA).

REFERENCES

- [1] Narges Mehran, Dragi Kimovski, and Radu Prodan. A two-sided matching model for data stream processing in the cloud – fog continuum. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 514–524. IEEE, 2021.
- [2] Dragi Kimovski, Narges Mehran, Christopher Emanuel Kerth, and Radu Prodan. Mobility-aware iot applications placement in the cloud edge continuum. *IEEE Transactions on Services Computing*, 2021.
- [3] Yue Zhou, Yue Yu, and Bo Ding. Towards mlops: A case study of ml pipeline platform. In *International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*, pages 494–500. IEEE, 2020.
- [4] Ivan Lujic, Vincenzo De Maio, Sri Kumar Venugopal, and Ivona Brandic. Sea-leap: Self-adaptive and locality-aware edge analytics placement. *IEEE Transactions on Services Computing*, 2021.
- [5] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.
- [6] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.
- [7] Seyed Yahya Nikouei, Yu Chen, Sejun Song, Ronghua Xu, Baek-Young Choi, and Timothy Faughnan. Smart surveillance as an edge network service: From harr-cascade, svm to a lightweight cnn. In *2018 IEEE 4th international conference on collaboration and internet computing (cic)*, pages 256–265. IEEE, 2018.
- [8] Cunqian Yu, Bin Lin, Ping Guo, Wei Zhang, Sen Li, and Rongxi He. Deployment and dimensioning of fog computing-based internet of vehicle infrastructure for autonomous driving. *IEEE Internet of Things Journal*, 6(1):149–160, 2018.
- [9] Tarek Menouer. Kcss: Kubernetes container scheduling strategy. *The Journal of Supercomputing*, 77(5):4267–4293, 2021.
- [10] Redowan Mahmud, Samodha Pallewatta, Mohammad Goudarzi, and Rajkumar Buyya. Ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments. *arXiv preprint arXiv:2109.05636*, 2021.
- [11] Zahra Najafabadi Samani, Nishant Saurabh, and Radu Prodan. Multilayer resource-aware partitioning for fog application placement. In *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*, pages 9–18. IEEE, 2021.
- [12] Zahra Pooranian, Mohammad Shojafar, Paola G Vinuesa Naranjo, Luca Chiaraviglio, and Mauro Conti. A novel distributed fog-based networked architecture to preserve energy in fog data centers. In *Mobile Ad Hoc and Sensor Systems (MASS), 2017 IEEE 14th International Conference on*, pages 604–609. IEEE, 2017.
- [13] Changsheng You, Yong Zeng, Rui Zhang, and Kaibin Huang. Asynchronous mobile-edge computation offloading: Energy-efficient resource management. *IEEE transactions on wireless communications*, 17(11):7590–7605, 2018.
- [14] Ganesh Ananthanarayanan, Victor Bahl, Landon Cox, Alex Crown, Shadi Noghahi, and Yuanhao Shu. Video analytics-killer app for edge computing. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 695–696, 2019.
- [15] Nikolay Nikolov, Yared Dejene Dessalk, Akif Quddus Khan, Ahmet Soylu, Mihail Matskin, Amir H Payberah, and Dumitru Roman. Conceptualization and scalable execution of big data workflows using domain-specific languages and software containers. *Internet of Things*, page 100440, 2021.
- [16] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 412–426, 2021.
- [17] Narges Mehran, Zahra Najafabadi Samani, Dragi Kimovski, and Radu Prodan. Matching-based scheduling of asynchronous data processing workflows on the computing continuum. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 58–70, 2022.
- [18] Henry Wilde, Vincent Knight, and Jonathan Gillard. Matching: A python library for solving matching games. *Journal of Open Source Software*, 5(48):2169, 2020.
- [19] Michael Chima Ogbuachi, Anna Reale, Péter Suskovics, and Benedek Kovács. Context-aware kubernetes scheduler for edge-native applications on 5g. *Journal of Communications Software and Systems*, 16(1):85–94, 2020.
- [20] Mulugeta Tamiru, Guillaume Pierre, Johan Tordsson, and Erik Elmroth. mck8s: An orchestration platform for geo-distributed multi-cluster environments. In *ICCCN 2021-30th International Conference on Computer Communications and Networks*, 2021.
- [21] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. Cloud, fog or edge: Where to compute? *IEEE Internet Computing*, 2021.
- [22] Siniša Segvić, Karla Brkić, Zoran Kalafatić, and Axel Pinz. Exploiting temporal and spatial constraints in traffic sign detection from a moving vehicle. *Machine vision and applications*, 25(3):649–665, 2014.
- [23] Lina Al-Kanj, Wissam El-Beaino, Ahmad M El-Hajj, and Zaher Dawy. Optimized joint cell planning and bs on/off switching for lte networks. *Wireless Communications and Mobile Computing*, 16(12):1537–1555, 2016.
- [24] Soufiane Dahmani, Mohammed Gabli, and Abdelhafid Serghini. A green fuzzy multi-objective approach to the rnp problem for lte networks. *Progress in Artificial Intelligence*, pages 1–13, 2021.