Received 13 October 2025, accepted 24 November 2025, date of publication 26 November 2025, date of current version 5 December 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3637510

RESEARCH ARTICLE

Impact Analysis of Algorithm Optimization on Robot Deep Learning Localization Model for Real-Time Execution

CÉDRIC MELANÇON[1], (Member, IEEE), MAAROUF SAAD[1], (Senior Member, IEEE), KULJEET KAUR[1,2,3], (Member, IEEE), AND JULIEN GASCON-SAMSON[4], (Member, IEEE)

[1]Department of Electrical Engineering, École de technologie supérieure, Montréal, QC H3C 1K3, Canada
[2]Department of Electrical Engineering, Canadian University Dubai, Dubai, United Arab Emirates
[3]Centre for Research Impact and Outcome, Chitkara University Institute of Engineering and Technology, Chitkara University, Rajpura, Punjab 140401, India
[4]Department of Software and IT Engineering, École de technologie supérieure, Montréal, QC H3C 1K3, Canada

Corresponding author: Cédric Melançon (cedric.melancon.1@ens.etsmtl.ca)

ABSTRACT The increased demand for autonomous robots in industries such as healthcare, manufacturing, and logistics is driven by the need to address labor shortages and enhance operational efficiency. A critical aspect of these robots is their ability to navigate complex environments autonomously, which relies heavily on multiple artificial intelligence models, including visual odometry. Visual odometry enables robots to estimate their motion by analyzing visual data, making it essential for navigation and obstacle avoidance. However, ensuring such models operate within real-time constraints is paramount for the robot's responsiveness, safety, and overall functionality. The complexity and depth of such a model, which utilizes camera images and other sensor data, make it challenging to achieve real-time performance. Thus, this paper thoroughly evaluates the performance of a deep-learning visual odometry model, with a focus on its execution time and computational overhead. Furthermore, it proposes an optimized implementation to meet the stringent real-time requirements for safe and efficient robot operation. The optimization is achieved through algorithm improvement to preserve the model structure and accuracy. Multiple hardware platforms were utilized to demonstrate the resource differences between edge, fog, and cloud deployments, thereby validating the effect of the proposed model optimization. The resource usage is also compared to see the impact of the modification on multiple aspects, including computation, memory, temperature, and energy. The resulting model operationalization improves the execution time while highlighting limitations for some hardware configurations.

INDEX TERMS Algorithm, artificial intelligence, autonomous robots, energy consumption, optimization, parallel processing, real-time.

I. INTRODUCTION

Over the last decades, robotics has shown promising evolution, and autonomous robotics solutions are increasingly present in our lives. Many industrial fields benefit from the use of such robots. In construction, robots are used to install ceiling boards, as described in [1], which is a very demanding task for humans due to the weight of the boards and the necessity of working at heights. The industrial sector also uses autonomous robots to transport components from one workstation to another. Authors in [2] propose a robot framework that enables tasks such as 3D printing parts while navigating to optimize production. Robots can also be used in agriculture to identify, localize, and pick fruits as detailed in [3]. Identifying potential diseases in the orchard and sorting out problematic fruits is possible with artificial intelligence (AI) [4]. Finally, healthcare is another field that has started to leverage robotics. Authors in [5] explained the adoption of robotics in the healthcare sector to perform various roles, such as frontline workers, logistics, escort, manufacturing, and surveillance.

Different AI models can be leveraged to perform such industrial applications, but they require significant computing power. However, running all computational tasks locally

The associate editor coordinating the review of this manuscript and approving it for publication was Mueen Uddin.

on the robot increases its cost manifold, primarily due to the miniaturization of the electronic components that must fit within the robot's body. Moreover, the miniaturization of electronics increases fabrication complexity and the likelihood of thermal issues. Additionally, such computing power is highly energy-intensive, which poses a problem for battery-operated robots [6], [7]. To circumvent these challenges with hosting computationally intensive tasks on robots, the *Internet of Robotic Things (IoRT)* has emerged as a potential solution. As introduced in [8], IoRT lays the foundations for distributed robotics, enabling it to benefit from external data sources and enhance the robotics solution. It uses cloud computing technologies to provide distributed computing capabilities for data from Internet of Things (IoT) sensors. For instance, authors in [9] proposed applying this concept to monitor the health of indoor plants by integrating environmental sensors with robot observations.

The use of cloud resources implies a high communication latency, as stated in [10]. Many algorithms or models require real-time constraints to ensure the safety of autonomous agents, which is not compatible with high-latency solutions. Traffic sign detection [11], navigation [12], and control [13] are examples of features that AI models solve, necessitating fast responses. Authors in [14] compared the accuracy of various implementations of the You Only Look Once (YOLO) model. It can be stated that larger architectures generally achieve higher accuracy but at the cost of longer execution times. Deploying such a model in a real-time, time-critical system requires carefully balancing execution time and accuracy. Furthermore, larger models, characterized by greater depth and a higher number of neurons per layer, require more computational resources, resulting in increased energy consumption.

Image processing models are good examples of such large models. Processing camera feeds is computationally intensive and plays a crucial role in various functionalities of autonomous agents, including robots and vehicles. It is used for positioning and movement control [15] or obstacle detection [16]. For instance, the Simultaneous Localization and Mapping (SLAM) algorithm has been used to explore an environment to build a map that can be used for robot navigation [17], [18]. Visual odometry is a feature used in robotics to estimate the pose and localization of the robot using image processing. Authors in [19] proposed a Deep Learning (DL) approach for visual odometry through sensor fusion of Inertial Measurement Unit (IMU) with color camera images. Other sensors can also be used for indoor localization, such as Ultra-Wideband (UWB) sensors [20] or Light Detection and Ranging (LIDAR) sensors [21]. The resulting localization is also used for robot motion through motor control. These time-critical tasks ensure the robot avoids obstacles or stops when they are detected. All these robotics features utilize multiple sensors that generate a large volume of data, presenting a challenge for real-time processing.

Most of the algorithms mentioned above are implemented for the Robot Operating System (ROS) middleware [22], a widely used framework in robotics solutions. It offers a distributed approach to performing tasks, with sensor nodes providing telemetry data and controller nodes utilizing sensor data to perform tasks such as motor control, localization, navigation, and more. The proposed work utilizes ROS to execute an AI model on sensor data input, leveraging all the features available in the framework.

A. MOTIVATION

This paper aims to execute a resource-demanding AI model, such as DL-based Visual Odometry, into a robotics solution with real-time constraints without impacting the model's accuracy. The execution time improvement is achieved through algorithm modification, rather than modifying the model size, which can affect its accuracy. The goal is to keep the model structure and parameter values of a proven working DL model to improve execution time and meet real-time constraints.

Knowing precisely a robot's three-dimensional position and orientation is the foundation of any robotics solution. These values can be used through derivatives to obtain the robot's linear and angular velocities or accelerations with the second-order derivative. The obtained results are used for robot control and navigation. The execution rate of such algorithms depends on multiple factors, including the robot's speed, responsiveness, and accuracy. The faster the rate, the faster the robot can react and the more stable its motion will be. When dealing with kinematic control (based on position, velocity, and acceleration), the required execution rate is slower than that of dynamic controllers (using wheel torque, friction, and all dynamic aspects of the robot), as specified in [23].

Dynamic controllers are more sensitive to variations and need faster reactivity to ensure smooth motion. Such controllers are more flexible but more complex to model and implement, and require more computing power than kinematic controllers. For kinematic control, increasing the execution rate increases responsiveness and accuracy, but also increases computational needs. If the robot's speed of operation is slow, the need for responsiveness is lower. The rate for slow-moving robots (0.1 to 0.5 m/s) can be 10-20 Hz [24], while it can go up to 100 Hz [25] for fast-moving robots (higher than 1.0 m/s). Given that an autonomous robot navigating a crowded environment cannot move swiftly, we consider a maximum velocity of 0.75 m/s, which justifies a rate of 30 Hz, representing the real-time constraint for the visual odometry model execution.

This paper leverages an existing DL model for robot localization proposed in [26] and available on GitHub, which serves as the baseline for this work. It uses data from the IMU and color camera images. Since image processing is the model's most time-consuming component, the image rate determines the model's execution speed. Therefore, the

TABLE 1. Real-time model references.

Use case	Reference	Model type	Methodology	Timing
Traffic sign detection	[27]	CNN	Hyperparameter tuning	Not measured
Traffic sign detection	[28]	CNN	Region of interest segmentation	Not measured
Traffic monitoring	[29]	Sparse CNN	Model optimization with sparse network	50% reduction
Facial recognition	[30]	CNN	Various model comparison	18.1 ms/image
Intrusion detection	[31]	LSTM, CNN, MLP	Model architecture comparison	14.29 ms/image
Human Pose Estimation	[32]	CNN	Model comparison	Not measured
Crowd enumeration	[33]	CNN	Model modification	Not measured
Collision detection	[16]	CNN	Execution pipeline	1.52 s/test
Prostate cancer detection	[34]	Transformers	Model modification	993 queries per second
Robot Simultaneous Localization and Mapping (SLAM)	[35]	CNN	Pruning and quantization	Not measured
Robot Control	[36]	CNN	Pruning, quantization, fine-tuning and clustering	15-79 Hz

selected camera frame rate is 30 Hz, and the model is applied to every received camera image.

B. RELATED WORK

AI processes are divided into two operations, namely training and execution (or inference). Both operations are not necessarily intended to be performed on the same computer configuration, as training typically requires more memory and computing resources to determine the best parameters for achieving the desired accuracy using a large amount of data from a given dataset. The resulting network parameters are used to execute the model inference on a single set of inputs.

Table 1 presents multiple references to real-time scenarios solved through AI models. It is essential to note that while authors discuss real-time constraints for AI models, they refrain from precisely quantifying the timing objectives. It becomes difficult to assess if the resulting timings claimed (when provided) are valid for such a real-time scenario. The use cases necessitating real-time execution include autonomous vehicle scenarios (such as collision detection, traffic sign detection, intrusion detection, and traffic monitoring), crowd-related scenarios (facial recognition, human pose estimation, and crowd enumeration), and healthcare applications (such as prostate cancer detection). The implemented models utilize well-known neural networks, including Convolutional Neural Networks (CNN), Multilayer Perceptrons (MLP), Recurrent Neural Networks (RNN), such as Long Short-Term Memory (LSTM), and transformers. The methodology for achieving the desired execution time involves hyperparameter tuning, utilizing sparse networks, comparing multiple model architectures to select the optimal one, and integrating model execution into a pipeline to process model predictions. Another approach is to divide the full-scaled image into smaller images to reduce the model size and increase the execution time. It is also possible to parallelize multiple regions when multiple GPUs are available. The timing analysis depends on the model's usage and the type of input. All of these works utilize color images captured by a camera, except for the prostate cancer detection scenario, which employs ultrasound images. The latter advertises fast model processing, which can be attributed to the grayscale image and the small model size.

Pruning and quantization techniques are discussed in [35]. The authors successfully optimized the SLAM model by up to 79.8% while keeping its accuracy by retraining it with fine-tuning techniques. Authors of [36] compared various model optimization techniques, including pruning, quantization, fine-tuning, and clustering, across different AI frameworks, such as PyTorch and TensorFlow, as well as their optimized variants. Their analysis reveals that, depending on the framework, specific optimization techniques are more effective than others, yet can significantly improve inference time with minimal impact on model accuracy, and sometimes even yield better results.

Another essential factor for successful autonomous robotics solutions is their energy efficiency [37]. Since the robots are battery-operated, particular care must be taken to minimize the computing and memory requirements of the designed algorithm. An analysis of the impact of AI on energy consumption is detailed in [38], and the obtained results highlight that model choice significantly impacts energy consumption (from basic machine learning algorithms to complex neural networks). Further, the programming language used for implementation can also influence energy consumption. Thus, particular attention should be devoted to selected models that are reliable in accuracy and energy-efficient.

Another essential factor impacting the robot's energy consumption is the resource type chosen for the model execution, *i.e.*, Graphics Processing Unit (GPU) and Central Processing Unit (CPU). Authors in [39] compared the energy consumption of training and inference execution on GPU and CPU. Even if the GPU consumes significantly more energy than the CPU, the gain in execution time makes it more energy efficient than the CPU for a particular task. They also highlighted that a GPU consumes significant power even when idle. This means that when using GPU-based systems for non-AI workloads, the energy consumed by idling GPUs considerably impacts energy efficiency. A model partitioning solution is proposed in [40], which distributes the model on GPU and CPU for edge devices to optimize the balance of energy consumption and execution time.

Different approaches are discussed in the literature to optimize AI models that reduce energy consumption, including bagging algorithms, pipelining, and dynamic programming. Authors in [41] presented a bagging algorithm to parallelize computation using mini-batches, thereby improving energy consumption while minimizing the impact on model performance. Josyula et al. [42] proposed a pipeline solution to perform point cloud segmentation to optimize run-time execution. A downside of pipelining is the addition of communication overhead, which affects the overall execution time. Additionally, the segmentation results in data reduction, which improves execution time. Li et al. [43] proposed a dynamic programming approach to distribute processes on both GPU and CPU on mobile devices. Their results depicted better efficiency than using only a GPU. Along similar lines, authors in [44] explored compiler optimization to improve model execution time, demonstrating faster execution times on both CPU and GPU while having a minimal impact on model accuracy.

Unlike the optimization techniques discussed, which focus primarily on model architecture changes or hyperparameter tuning, this paper's proposed approach preserves the model structure and, instead, optimizes the execution pipeline and deployment strategy through algorithmic optimizations.

C. CONTRIBUTIONS

Since the work relies on a real-world scenario based on visual odometry, this paper defines the execution time limits and proposes an optimized model to meet these constraints. The main contributions of this paper are:

- *Definition of the real-time constraint* with justifications of an autonomous robot localization module.
- *Multiple phase algorithm optimization* of a Visual Odometry DL model, including the adaptation of the model in ROS.
- *Impact analysis* of the various phase optimizations on the execution time, resource usage, and energy consumption.
- *Heterogeneous hardware configuration comparison* to model the execution at different layers (edge, fog, or cloud computing).

It is important to emphasize that the structure of the Visual Odometry DL model was intentionally left unchanged in this study. While some basic hyperparameter tuning was performed to achieve reasonable accuracy, our primary objective was to evaluate and optimize the model's execution pipeline for real-time performance, rather than to maximize accuracy through extensive hyperparameter tuning. The goal is to establish a model baseline for evaluating the impact of our proposed work on inference execution time, aiming to achieve a 30 Hz frame rate. Further work will be conducted in the future to enhance the model's accuracy and, by applying the proposed algorithm, to meet both objectives of accuracy and real-time processing.

D. ORGANIZATION

This paper is organized as follows. Section I-B provides an overview of the related work. Section II introduces the baseline visual odometry model used for this work and explains the process used for the AI model training to obtain the results for this paper. Section III explains the various optimization phases applied to the model to operationalize it and meet real-time criteria. Section IV evaluates the performance of the optimized model and provides an analysis of the resources used for execution, along with a discussion of the various results. Finally, a conclusion presents potential areas for improvement in future work.

II. VISUAL ODOMETRY MODEL

The concept of odometry uses data from various sensors to estimate the position and orientation variation of the robot. By itself, it does not directly provide the robot's localization. Still, when the original robot's position is known, the global position can be computed by applying the relative motion to the previous localization.

Visual odometry provides the same relative motion but utilizes visual data to enhance the accuracy. This data can originate from sensors such as camera feed (color or depth images) and LIDAR point cloud. The high-level structure of the visual odometry model, proposed in [26] and used as the baseline for this work, is detailed in Fig. 1. It performs sensor data fusion using data from an IMU and an RGB Camera to predict the robot's relative position and orientation. This predicted localization can feed other services, such as motor control or navigation models.

This model is divided into multiple submodels. The first submodel is responsible for the feature extraction using the Flownet2S model introduced by [45]. Flownet is a DL-based optical flow algorithm that takes two frame images of a movement and detects the motion variation from one frame to another. Since the visual odometry model uses the Flownet only for feature extraction, the last dense layer from the model is removed. If the camera frame rate is 30 Hz, the first image captured does not produce any output data; however, the subsequent frames are used in pairs with the previous frame as input to the feature extraction submodel.

The resulting data, labeled *Observation*, is pushed into the next submodel, acting as an encoder. The encoder is detailed in Fig. 2. It is designed to help reduce the data size used for the remaining model through three dense layers. The resulting data reduction is labeled *Encoded Observation*.

The transition submodel following the encoder is represented in Fig. 3. It is responsible for the sensor fusion using the *Encoded Observation* and the data originating from the IMU sensor. A sensor rate of 100 Hz is considered for the model design, meaning that three IMU frames are received between two camera frames. This data is processed using Gated Recurrent Units (GRU) RNN [46]. The output of the RNN is provided to a dense layer to scale the result data size and is then concatenated with the encoded image before being used in an LSTM [47]. The idea behind the two RNNs is to leverage the previous data from the sequence over time. This is performed through a closed loop, and the resulting RNN hidden layer data is reinjected in the subsequent execution as

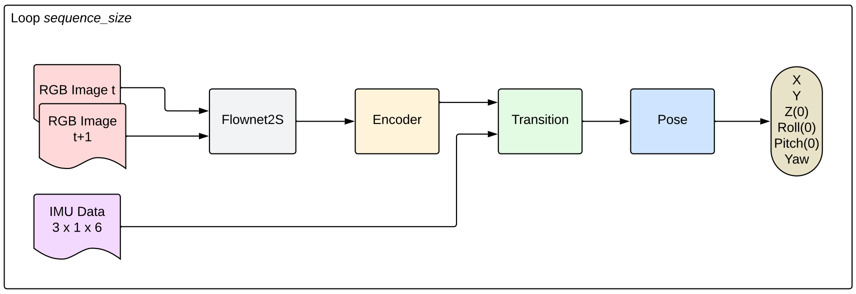

FIGURE 1. Deep learning visual odometry model.

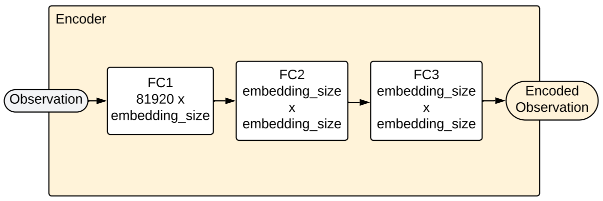
Encoder
Observation
FC1 81920 x embedding_size
FC2 embedding_size x embedding_size
FC3 embedding_size x embedding_size
Encoded Observation

FIGURE 2. Encoder model.

a memory of the past. Two fully connected layers follow to provide features resulting from this sensor fusion, labeled *Out Features*.

Finally, a pose model, as described in Fig. 4, performs pose prediction through a succession of dense layers. They are divided into two distinct paths. The first path (in red) predicts the position in two dimensions (since the robot is moving on flat floors, the Z coordinate is considered to be zero) with a minor influence from the angular portion on the Y coordinate. The second path (in green) predicts the yaw (rotation around the Z-axis) while the roll (X) and pitch (Y) are considered zero due to the 2D motion and to reduce the complexity. The result is the relative variation of the position/orientation between two frames.

The obtained relative position and orientation are converted into a transform matrix and used to determine the new predicted robot's global position. It is expected that the initial position has to be known for this algorithm to work.

A. DEPLOYMENT

Over the past few years, the approach to deploying robotics solutions has undergone significant evolution. Initially, the computation was only located on the robot platform. With the increase of AI models to execute robot operations, such a solution has limitations regarding available resources and energy. The concept of Cloud Robotics, as introduced in [48], addresses resource availability but introduces significant communication latency, rendering it ineffective for real-time scenarios. Resources used in the cloud can be virtualized to match computing requirements and scaled as demand grows.

Fog Robotics [49] brings the computation closer to the user, reducing the latency and making it more accessible for critical scenarios. The amount of resources is more limited than the cloud, but it can include computers with high-end resources. A compromise can be made using Edge Robotics, which brings computing power to the edge of the robotics solution with viable communication latency for real-time constraints. While Edge computing holds more resources than robots, it is limited compared with Fog capabilities.

This paper compares the various hardware configurations representing the deployment types (Edge, Fog, and Cloud).

B. MODEL TRAINING

The project's use case is based on indoor mobile robotics. Accordingly, the MIT Stata Center Dataset [50] was selected for training the model. The dataset is derived from an indoor scenario involving a robot equipped with components similar to those available in our lab. It also has the advantage of providing the data in a ROS bag file representing a real-time recording of a sequence. This sequence can be replayed in ROS to facilitate the inference evaluation in a near-reality environment. Specific ground-truth labeling is available for multiple sequences, facilitating supervised training. The data rate for the IMU is 100 Hz, and the camera feed runs at 30 Hz, as designed in the model.

The accuracy obtained through the model's training is not optimal, with an accumulated error of around 20 cm in X, 8 m in Y, and 20 degrees in theta. The primary factor contributing to the accumulated error in Y is the orientation error. While more extensive hyperparameter tuning could reduce these errors, such optimization was beyond the scope of this study, which focused on improving the execution pipeline and, therefore, the operationalization of the model, rather than refining the model. Another factor that can impact the model's accuracy is the image size. While the MIT Stata Center dataset provides 640×480 images, the original model was trained on the KITTI dataset [51] with 1382×512 images, resulting in more extensive training data.

While our experiments focus on the MIT Stata Center dataset due to its compatibility with our hardware and ROS-based pipeline, the operationalization strategies proposed are applicable to other datasets, such as KITTI (outdoor vehicle scenario) and EuRoC (three-dimensional aerial vehicle), which are used for training in the original paper. Each dataset

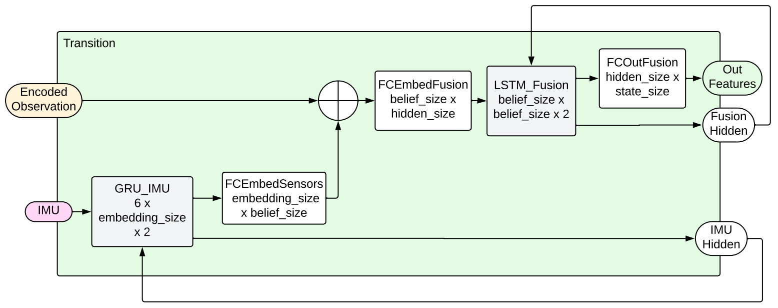
Transition
Encoded Observation
IMU
GRU_IMU 6 x embedding_size x 2
FCEmbedSensors embedding_size x belief_size
FCEmbedFusion belief_size x hidden_size
LSTM_Fusion belief_size x belief_size x 2
FCOutFusion hidden_size x state_size
Out Features
Fusion Hidden
IMU Hidden

FIGURE 3. Transition model.

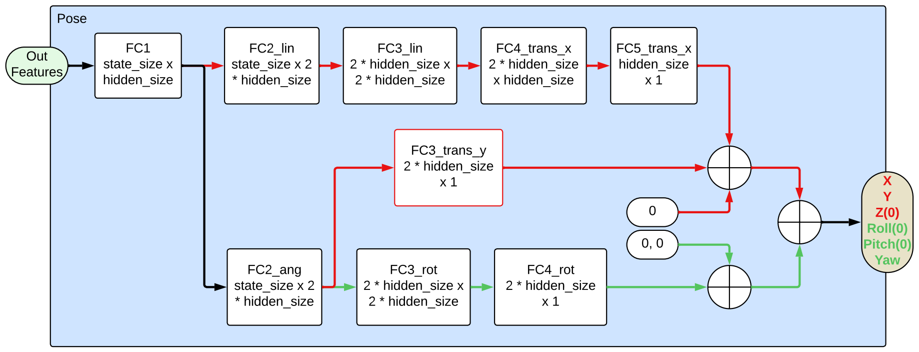
Pose
Out Features
FC1 state_size x hidden_size
FC2_lin state_size x 2 * hidden_size
FC3_lin 2 * hidden_size x 2 * hidden_size
FC4_trans_x 2 * hidden_size x hidden_size
FC5_trans_x hidden_size x 1
FC3_trans_y 2 * hidden_size x 1
0
0, 0
FC2_ang state_size x 2 * hidden_size
FC3_rot 2 * hidden_size x 2 * hidden_size
FC4_rot 2 * hidden_size x 1
X
Y
Z(0)
Roll(0)
Pitch(0)
Yaw

FIGURE 4. Pose model.

presents its own particularities in terms of vehicle velocity, indoor or outdoor environment, or two or three-dimensional navigation. They all provide sequences with camera images and IMU data. Since the aim of the project is indoor two-dimensional navigation, the characteristics of the MIT Stata Center dataset are more suitable, and no generalization is necessary for other scenario types.

III. MODEL OPERATIONALIZATION

The model's operationalization was performed in three phases. This section details them and discusses their potential impact on the model's execution time.

A. PHASE 1: ROS CONVERSION

The code baseline, implemented in Python with the PyTorch framework, is incompatible with ROS. The first phase involves converting the visual odometry model to make it compatible with the ROS framework. A ROS node can combine the concepts of publishers sending data, subscribers receiving data, and services performing client-server actions. This conversion is mainly necessary to leverage the replay capability provided by the ROS bag file. Another motivation for this phase is that ROS is widely used in the robotics industry. This phase involves no changes to the original model execution, but rather the method for providing the dataset to the model (from a regular PyTorch dataset to a ROS subscriber). The ROS2 Humble distribution was used for this work.

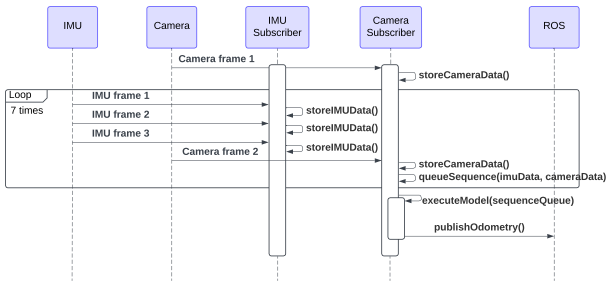

FIGURE 5. Phase 1 sequence diagram.

The resulting sequence diagram is represented in Fig. 5. Each sensor stream (IMU and RGB Camera) is handled by a dedicated ROS subscriber running in separate threads. Every IMU and Camera data is stored in a queue in a loop over seven frames (*sequence_size*). Mutexes are used to synchronize access to the shared 7-frame queue, ensuring that only one thread can modify the queue at a time. When an image is received, the process copies the last three IMU values and the received camera image into a sequence queue. The whole model can be performed when the queue has seven sequence frames. A mutex also protects the model execution to prevent concurrent predictions. The model is implemented precisely as described in the base code, which computes the prediction by looping over seven frames, performing the Flownet, Encoder, Transition, and Pose submodels for each sequence execution, as represented in Fig. 1.

ROS can be executed in a single-threaded or multi-threaded fashion. The Phase 1 optimization leverages multi-threading to perform every message reception in a separate thread. ROS also offers two options for thread execution, namely, mutually exclusive and reentrant execution [52]. The concept of mutually exclusive execution is similar to protecting the thread with a mutex, ensuring that a thread completes its execution before starting the execution of the following thread instance. Reentrant execution enables the parallel execution of processes, leaving the developer responsible for ensuring thread-safe execution. The latter option is implemented for this optimization phase to simplify the implementation of the subsequent improvements. This implementation is very similar to the mutually exclusive thread execution, except that storing camera image data can be executed simultaneously with the model execution.

Since the model execution is protected, an overrun (execution time over the 30 Hz framerate) has a direct impact on the validity of the data. The synchronization between the RGB camera and the IMU would be broken, resulting in invalid predictions, which is considered a failure.

B. PHASE 2: PARALLELIZATION

The idea behind Phase 2 optimization is to remove the Flownet submodel from the overall model execution, allowing for parallel processing. While in the previous phase, we introduced multithreading, its only benefit is that it enables sensor data acquisition (since a mutex protects the execution of the whole model). The mutex acquisition difference between phases 1 and 2 is represented in Fig. 6. The fact that the mutex acquisition in phase 2 is pushed after the Flownet submodel allows it to be executed in parallel to the remaining model execution.

Instead of waiting for the entire visual odometry model to be completed, the Flownet submodel execution is initiated by the ROS subscriber upon receiving an image, using the two previously received frames. If the odometry prediction comes before the end of the Flownet execution, the new prediction will start. Otherwise, it will wait for the process to be completed. The main reason this modification is possible is that Flownet is completely decoupled from the remainder of the model, focusing solely on camera images.

This simple modification can significantly improve execution time due to parallelization, but it is expected to consume more resources for the same reason.

For this phase, an overrun of the Flownet execution would break the synchronization with the IMU, meaning that the three latest received IMU frames would not be aligned with the pair of camera images, which would be considered a failure.

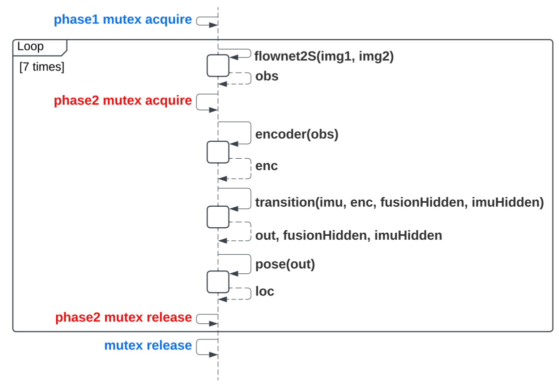

FIGURE 6. Model execution sequence diagram.

C. PHASE 3: LOOP UNROLLING

The Phase 3 implementation continues to execute the visual odometry model over seven sequence frames at each camera frame reception. The loop unrolling optimization aims to leverage the RNN concept appropriately. When the GRU or LSTM network is executed, it provides the resulting tensor and the values of its hidden layers, representing its internal memory. The values of these hidden layers can be injected as input for the subsequent execution, meaning that there is no added value to computing the result over the entire sequence, since only the last values are used. Sequence is essential for RNN, mainly for the training process. The initial hidden layer values can be randomized or set to zero, and this initialization is considered a hyperparameter for the training process.

To improve the execution time, this loop is unrolled, meaning the hidden layers (*Fusion Hidden* and *IMU Hidden* in the transition model) are stored for the subsequent model execution. The sequence diagram in Fig. 7 illustrates the execution of the visual odometry model, including the loop unrolling optimization. The thread synchronization mutexes are kept based on the Phase 2 implementation to ensure parallelization of the FlowNet submodel.

Before obtaining the first result from the model, Flownet2S requires two frames. Also, since the hidden frames of the RNN are either initialized to random values or zeros, the first predictions are not very accurate. To mitigate this problem, a warm-up period of 7 frames is considered. Therefore, the first localization data will be available only after the 8^{th} frame. To emphasize the different execution philosophy compared with Phase 2, the model execution is labeled *stepModel*, meaning that the process performs one frame step instead of the whole seven-frame execution.

The benefit of this phase is the reduction of computations during model execution, as the sequence loop of 7 frames is accumulated over time instead of being executed at every camera frame reception. This reduction removes some burden on processing units (CPU/GPU) but does not affect memory usage.

The Flownet overrun impact on the synchronization is the same as for Phase 2, which is considered a failure.

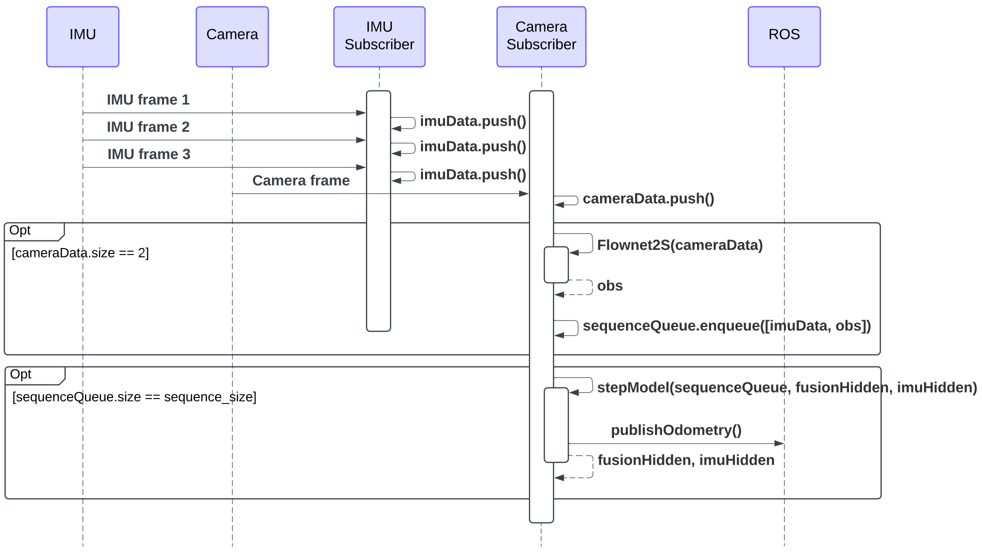

FIGURE 7. Phase 3 sequence diagram.

IV. EVALUATION

To validate the model in various deployment scenarios (edge, fog, and cloud), different hardware configurations are used, which are detailed in Section IV-A along with a description of the executed tests. Section IV-B discusses the frame rate validity considered for the evaluation, while Sections IV-C, IV-D, and IV-E advertise the phase-by-phase results to show the improvement brought by each implementation in terms of execution time. To further analyze the results, resource usage is evaluated in Section IV-F, along with power consumption and hardware temperature in Section IV-G. Finally, Section IV-H concludes the evaluation with a discussion of the impact of the optimization changes on model execution.

A. METHODOLOGY

Three hardware configurations were used to compare the performance of the model's execution (described above) for robot localization. These three configurations vary in terms of resources and are used to represent different execution scenarios in a distributed robotics solution. Typically, servers at the fog and cloud levels utilize compute virtualization; however, we consider the suggested hardware as a representative configuration for such virtual machines. These are detailed as follows:

- **Jetson**: A Jetson Orin AGX 32 Gb RAM running over Jetpack 6.0 (based on Ubuntu 22.04 Desktop), representing the robot's computing power at the edge.
- **Laptop**: A laptop with an Intel Core i7-10750H, 32 Gb of RAM, and an NVIDIA GTX 1660 Ti (6 Gb of GPU memory) running over Ubuntu 22.04 Desktop, which is considered as an edge node (not localized on the robot but could be installed as a gateway on a floor) or a small fog node.
- **Desktop**: A desktop computer with an Intel Core i7-12700K, 32 Gb of RAM, and an NVIDIA RTX 3090 (24 Gb of GPU memory) running over Ubuntu 22.04 Desktop, considered a fog or cloud node.

For all configurations, the ROS package running the visual odometry model is executed in a Docker container using CUDA, and the sequence is also executed in a separate Docker container on the same computer. Communication between the two containers is done through the Docker virtual network and is not advertised outside the computer.

The sequence used from the MIT Stata Center dataset is 2012-04-03-07-56-24, which contains 45720 camera frames representing a run of 25 minutes and 27 seconds.

The evaluation examines the execution time for each optimization phase and the resource usage, including CPU, GPU, Random-Access Memory (RAM), power consumption, and temperature. The Basic Hardware Monitor script by [53] was integrated into the ROS node to obtain the metrics detailed above. To obtain GPU information on the Jetson, some changes were required using the `jetson_stats` library from [54]. The energy consumed during the sequence was monitored using a wattmeter to capture the overall energy consumption throughout the sequence execution. The computer is directly connected to the wattmeter to obtain the energy data, and the acquisition is reset manually when the sequence is initiated. In the following box plots (Figs. 8 to 9), the red line represents the maximum execution time of 33.3 ms (a frame rate of 30 Hz).

For each optimization phase, the entire dataset sequence was processed independently on each hardware configuration. All tests were conducted under identical conditions, with system load minimized and hardware monitoring scripts running concurrently to ensure consistency in resource usage

TABLE 2. Number of frames executed over the sequence by configuration and development phase.

Configuration	Phase 1	Phase 2	Phase 3
Jetson	5173	7065	30780
Laptop	4513	10411	45719
Desktop	45713	45713	45719

measurement. The random seed value used for all the runs is similar to ensure consistency. Therefore, the model accuracy is equivalent in all cases as depicted in Section II-B.

B. FRAME RATE

It is essential to note that failing to meet the 30 Hz threshold results in prediction failure. The model relies on two camera frames and three IMU frames to predict its position before the next camera frame is captured. If a deadline is missed, two options are available. First, the faulty frame could be ignored; however, in that case, it would impact the model, as it expects consistent hidden layer values over time for the RNN. The other option is to always wait for the prediction.

Table 2 represents the number of frames executed over the 25-minute sequence based on the second approach. Under ideal circumstances, we would expect 45,719 prediction values (one frame is not used for prediction due to the Flownet model). From these values, it is evident that the second option has an impact. An accumulation of time can be observed, indicating that services requiring localization will also be delayed. The laptop and desktop configurations work optimally for Phase 3, with all frames processed. Still, for the Jetson, only 67% of the frames are executed over the nominal sequence time. The result of this overflow makes it impractical for real-time applications since the prediction of the last frame would be computed approximately 12 minutes later. The implemented improvements have a significant impact. For the Jetson, during the nominal sequence time, only 11.3% of the frames were processed with Phase 1, 15.5% for Phase 2, and 67.3% for Phase 3. The laptop shows similar improvement with only 9.9% of the frames processed for Phase 1, 22.8% for Phase 2, and a successful 100% for Phase 3. The Desktop was already running almost fine with 99.99% of the frames processed for Phase 1, no noticeable improvement for Phase 2, and 100% for Phase 3.

C. PHASE 1

The results in Fig. 8 illustrate the execution time of the three configurations for the first optimization phase. This phase is considered the baseline since the original code was incompatible with ROS.

The box plot diagram shows that the Jetson's average execution time (248 ms) is better than that of the laptop (278.1 ms). The laptop configuration exhibits greater variability, ranging from 6 ms to 629 ms, compared to the Jetson's variability, which ranges from 14 ms to 540 ms. However, the real-time constraint of 30 Hz (or 33.3 ms per frame) is not respected with the laptop and the Jetson. The desktop configuration is usually within tolerances, with an average execution time of 10.7 ms, but some outlier values are taking longer to process (up to 117 ms).

To understand the impact of hardware configurations, comparing their parameters is mandatory. The Jetson Orin AGX features 1792 CUDA cores, compared to 1408 for the laptop's GTX 1660 Ti. Based on [55], its clock speed (930 MHz) is less than the laptop's (1530 MHz). The Jetson has a floating-point processing power of 3.333 TFLOPS compared with 5.027 TFLOPS for the computer GPU. When comparing these specifications, the GTX 1660 Ti presents better parameters but performs worse than the Jetson. One possible justification is that the GTX 1660 Ti is a general-purpose GPU, while the Jetson is explicitly designed for AI tasks. The Jetpack operating system, while based on Ubuntu 22.04, is customized for the particular embedded characteristics of the Jetson. This specialization can explain the better results. In comparison, the RTX 3090 desktop is powered by 10,496 CUDA cores running at 1.7 GHz, which significantly surpasses the two other configurations.

Another possible reason for the high variability in execution time is the kernel's task scheduler. Since other processes run on the computer, the available CPU and memory can be affected. Moreover, the three configurations run a desktop version of Ubuntu, so the user interface update and other services are workloads the operating system must handle. For the Jetson, the CPU and GPU share the same memory, which can add complexity to the operating system's resource management. This is not the case for traditional Nvidia GPUs, which have dedicated memory.

D. PHASE 2

The execution time for the second optimization phase is also shown in Fig. 8. All configurations exhibit a better average execution time, although the improvement is less significant for the Jetson configuration. Effectively, the Jetson achieves an average execution time of 225.5 ms, with peaks of up to 422 ms, while the laptop achieves an average execution time of 144.7 ms. The desktop has improved to 4.3 ms and exhibits less variability. The desktop outliers are less frequent than for Phase 1 but are still problematic (up to 262 ms). The variation in laptop configuration is more significant, with a maximum of 529 ms. This variation can be explained by the accumulation of parallel processing induced by the frame overflow that may overwhelm the system.

E. PHASE 3

The final optimization phase execution time results are shown in Fig. 8. Again, the global performances were improved significantly for the Jetson (78.2 ms with a peak of 547 ms) and laptop (11.6 ms with a peak of 299 ms) configurations. The Jetson's average execution time is enhanced by 147.3 ms, and for the laptop, by 133.1 ms. In both cases, the variability is also narrowed down. Unfortunately, Jetson's performance is still higher than the real-time constraint tolerance, but the

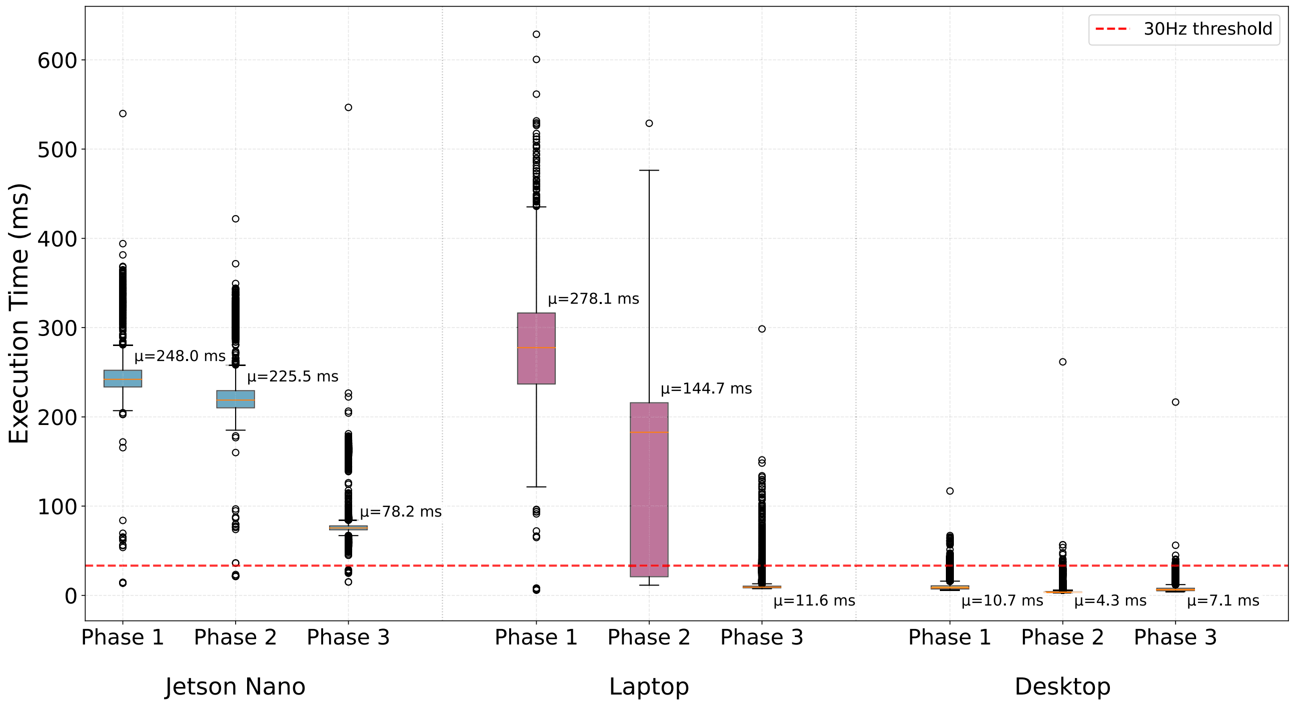

FIGURE 8. Model execution time for each configuration and phase.

TABLE 3. Submodels execution times (in ms) per configuration on development phase 3.

Configuration	Flownet	Encoder	Transition	Pose	Total
Jetson	48.2	19.4	10.6	17.4	95.6
Laptop	10.0	0.3	1.3	16.2	27.8
Desktop	6.1	0.2	1.4	1.4	9.2

laptop is now within the tolerance. However, the outliers can be problematic over time. The desktop average execution is slightly higher than Phase 2, with an average execution time of 7.1 ms, but with less variability and a peak value marginally better at 217 ms.

To understand the distribution of execution times across the entire visual odometry model, a breakdown of execution times for each submodel is presented in Table 3, which represents the execution time of each submodel for each configuration. The Flownet and Pose submodels are the most demanding due to the size of their neural networks. From these results, it is evident that running Flownet on the Jetson is not viable for meeting real-time constraints, as it exceeds the available time alone. The laptop and desktop timings are all below the 30 Hz (33 ms) threshold, as well as the sum of the submodel's execution time, which makes them viable configurations.

Suppose it is necessary to perform at least the FlowNet submodel due to the high latency induced by communicating images to other layers (edge, fog, or cloud). In that case, the model must be modified to reduce the size of the FlowNet network. This reduction will impact the accuracy,

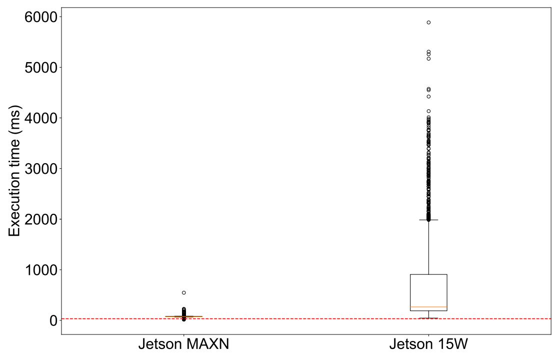

FIGURE 9. Phase 3 execution time for two Jetson power profiles.

necessitating some hyperparameter tuning to achieve a comparable model accuracy.

The Jetson Orin AGX offers multiple power configurations (15 W, 30 W, 50 W, and MAXN). These pre-settings try to limit the hardware's power consumption to the provided value. Although it is not guaranteed that the limits will not be exceeded, the average consumption is below the threshold. The MAXN configuration enables the hardware to utilize as much power as necessary, provided it remains within the safety limits of the hardware. The model was executed using two different Jetson system parameters to determine their impact on execution time. The Fig. 9 shows the differences in execution time for the MAXN and 15 W, which are the extremes in terms of configuration.

TABLE 4. Jetson power profile configuration.

Profile	CPU Freq. (MHz)	GPU Freq. (MHz)	CPU Cores	GPU Cores	RAM Freq. (MHz)
15W	1113.6	420.75	4	3	2133
MAXN	2201.6	1301	12	8	3200

TABLE 5. Resource usage of each configuration by development phase.

Configuration	Phase 1			Phase 2			Phase 3		
	CPU (%)	RAM (%)	GPU (%)	CPU (%)	RAM (%)	GPU (%)	CPU (%)	RAM (%)	GPU (%)
Jetson	14.8	69.3	68.1	21.7	69.4	69.0	20.8	71.4	71.7
Laptop	17.4	30.6	42.8	17.2	30.6	25.3	24.2	30.7	25.1
Desktop	7.7	36.6	8.0	8.7	37.6	12.0	9.4	39.2	8.9

Power usage has a significant impact on performance. For the MAXN configuration, the average execution time of 78 ms exhibits smaller variability (ranging from 15 ms to 547 ms) compared to the 15 W configuration, which has an average execution time of 674 ms with a variation ranging from 15 ms to 5.5 seconds. This can be explained by the various parameter changes required to reduce power, as detailed in Table 4, which implies a slower processing frequency that directly impacts performance. To achieve real-time execution time for AI models on a Jetson, a compromise must be made regarding energy consumption.

F. RESOURCE USAGE

Another critical factor that influences the execution time is resource availability. Table 5 shows the CPU, RAM, and GPU usage metrics over the sequence execution. The comparison of resource usage between configurations is not relevant since the amount of resources varies from one to another, as seen in Section IV-A. However, the resource usage for a given configuration from one development phase to another is interesting.

According to Table 5, the Jetson's CPU usage increases by 6.9% with the parallelization and slightly decreases by 0.9% for the loop unrolling. The laptop's CPU usage is stable for the first two phases and increases by 6.8% for Phase 3. The desktop displays a regular rise in CPU usage across the various phases. For the Jetson and the laptop, as the number of frames processed increases, the amount of disk access required to write the experimental data to files increases accordingly, which can justify the high variation. The CPU variation for the desktop configuration is almost linear and represents a 1.7% increase, which can be attributed to multiple factors external to the model execution. For the laptop and the Jetson, the increase can be associated with the rise in frames processed, which requires more processing for data preparation and publication.

RAM usage does not significantly vary from one phase to another. External factors could cause the variation. While all the hardware configurations share the same amount of memory (32 GB), the Jetson uses more memory than the other configurations. The justification stems from the fact that the Jetson memory is shared among the CPU and the GPU, whereas the GPU has its own dedicated memory for laptops and desktops.

The GPU usage on the Jetson presents a low increase of 5%. The behavior on the laptop and the desktop is entirely different. On the laptop, Phase 1 presents GPU usage almost 17.7% higher than the other phases. Despite the absence of parallel processing, the high GPU anomaly may be attributed to the sequential inference, which results in prolonged GPU engagement per frame. In contrast, later phases implement more efficient frame handling, allowing the GPU to return to lower usage states between computations. Additionally, the GTX 1080 GPU of the laptop differs significantly from the RTX 3090 and Jetson Orin AGX GPUs in terms of architecture, available resources, and driver-level resource management. All these factors can contribute to the observed anomaly.

On the desktop, Phase 2 presents a 1.2% increase. The increase can be justified by the parallelization of the Flownet with the remaining part of the model, which adds to the computational burden. Since only one step is performed during Phase 3 instead of the entire seven steps of the sequence, the GPU usage decreases from 12% to 8.9%. This effect is not visible on the laptop or the Jetson, mainly because of the accumulated execution delay, which puts constant pressure on the resources compared to the laptop and desktop, which benefit from the idle time between frames.

G. POWER AND TEMPERATURE

Since the visual odometry model is designed for a battery-powered robot, other factors, such as temperature and energy consumption, must also be analyzed. Table 6 presents the GPU temperature and the average power consumed for each configuration and development phase. The Jetson temperature is stable at 47.3°C from Phase 1 to Phase 2 and decreases to 45.9°C in Phase 3. The same temperature stability is observed from Phase 1 to Phase 2 for the laptop at 77.5°C with a slight variation of 0.9°C. The temperature rises to 83.7°C during Phase 3. For the desktop, the Phase 1 temperature is 54.6°C, which rises to 66.1°C for Phase 2 and decreases to 50.9°C for Phase 3.

Except for the laptop configuration in Phase 3 and the desktop configuration in Phase 2, there is no significant

TABLE 6. GPU Temperature in °C for each configuration and development phase.

Configuration	Phase 1	Phase 2	Phase 3
Jetson	47.3	47.3	45.9
Laptop	77.5	76.6	83.7
Desktop	54.6	66.1	50.9

TABLE 7. GPU Power in Watts for each configuration and development phase.

Configuration	Phase 1	Phase 2	Phase 3
Jetson	16.5	20.9	20.7
Laptop	28.2	38.5	47.0
Desktop	134.4	229.2	126.5

variation in temperature. The possible cause of the laptop's issue is a problem with the cooling system, as the main difference in resource usage is between the CPU and the GPU. For the desktop, we see a more significant usage of the GPU, which can explain the variation. Sustained GPU temperatures above 80°C may trigger thermal throttling, which reduces clock speeds and negatively impacts real-time performance. This behavior can explain the outlier execution time values of Phase 3 on the laptop configuration.

The GPU power usage is shown in Table 7. The **Jetson** consumes an average of 16.5 W for Phase 1 and is relatively stable at 20.9 W for Phase 2 and 20.7 W for Phase 3. The **laptop**'s power usage gradually increases from 28.2 W for Phase 1 to 38.5 W for Phase 2 and 47 W for Phase 3. The power consumption pattern for the **desktop** is different, with a power usage of 134.2 W in Phase 1, a significant increase to 229.2 W in Phase 2, and a lower value of 126.5 W in Phase 3.

The Jetson is optimized for low-power computation, as an event in the MAXN power mode requires a maximum of 21 W. The temperature and power usage of the laptop and desktop can be correlated. A potential justification for this is that when the GPU is heating, the power necessary for the cooling system seems to consume a significant amount of power to reduce the temperature.

The GPU power is not necessarily the best way to identify the model's real impact. Table 8 details the GPU power consumption while adding information about the energy consumed over the sequence execution. When considering the energy, it is noticeable that the optimization has a negative impact. The desktop consumes 39 Wh less with all the optimizations, while the laptop's consumed energy increases by 7 Wh, and the Jetson remains at the same energy level.

The decrease in energy consumption from the desktop can be justified by the reduction in computation due to the Phase 3 loop unrolling. The energy values are not representative of the laptop and Jetson configurations, as noted earlier, since the monitoring was conducted over a 25-minute sequence execution. Unfortunately, most configurations and phases did not execute the process over all the frames, which increasingly impacts actual energy consumption. Effectively, when examining the desktop configuration that executed the model over all frames, regardless of the development phase, we see that, contrary to expectations, the optimization has a positive impact on energy consumption. It is worth noting that the total power consumed by the Jetson and the laptop for the first two phases is not representative, as it only covers a portion of the total frame to process.

H. DISCUSSIONS

The Table 9 summarizes the impact on the model execution time through the various optimization phases.

While the improvement is not sufficient for the Jetson, it is evident that with an improvement of 68.47%, the changes are still insufficient to meet the real-time constraints. Still, it significantly improves it, so that when combined with other model optimization methods, such as pruning and quantization, it would be possible to meet the goal. The 95.83% improvement for the laptop makes it compatible with the real-time constraints. While the desktop was already running within the constraints, the 33.64% improvement leaves more room for other processes to run on the same computer.

Due to the size of the neural network, DL-based visual odometry models for predicting robot localization are highly resource-demanding. The proposed algorithm modifications significantly improve execution time, but based on the obtained results, real-time execution on a Jetson cannot be achieved. While slightly increasing the energy consumption on the desktop configuration, the main impact of the optimization is the reduction of outliers that result in overruns.

An interesting aspect arises from studying the energy consumption of Jetson. Even if the resource usage increases, the impact on power usage is negligible. This emphasizes that the Jetson architecture is optimized for systems running on batteries, even when running extensive AI models. We would need to reduce the camera frame rate to run the model effectively on this configuration, thereby providing more time for model execution. Lowering the robot's speed would allow this rate to decrease.

Compared to other optimization methods, the proposed solution does not modify the model structure, ensuring the same accuracy as long as the execution time meets the real-time frame rate. Adding more optimization would potentially decrease model accuracy while improving execution time.

Since the desktop is the only configuration capable of running all the frames, it is the best example for comparing the impact of the optimization phases. The desktop's average execution time is approximately 2.9 ms longer than phase 2. However, the less frequent and shorter outliers show improvement. It can be seen that the optimization has a low impact except for the reduction of outliers. One justification for this is the availability of better resources than the Jetson and the laptop. Even at Phase 1, the average model execution time was within the limits of real-time execution. To have

TABLE 8. Total power for each configuration and development phase.

Configuration	Phase 1			Phase 2			Phase 3		
	min (W)	max(W)	total (kWh)	min (W)	max(W)	total (kWh)	min (W)	max(W)	total (kWh)
Laptop	54.3	107.2	0.029	57.9	103.7	0.027	28.7	115.3	0.036
Desktop	266.8	313.5	0.124	200.7	239.2	0.092	199.9	230.1	0.088
Jetson MAXN	19.8	33.6	0.011	20.3	31.5	0.010	18.4	34.1	0.011
Jetson 15W	-	-	-	-	-	-	14.2	19.5	0.007

TABLE 9. Execution Time Improvement Across Optimization Phases.

Configuration	Phase 1 → 2	Phase 2 → 3	Phase 1 → 3
Jetson	9.07%	65.32%	68.47%
Laptop	47.97%	91.98%	95.83%
Desktop	59.81%	−65.12%	33.64%

a better impact, the optimization needs to move the model execution closer to or even below the limit, as it frees resources induced by heavy parallelization (more than two frames executed at once).

Although our evaluation focuses on single-machine deployments, network latency and communication overhead are critical factors in distributed robotics. The network communication latency is greatly impacted by the data size and the data rate, as concluded in a previous work [56]. The image processing should preferably be executed on the robot itself to reduce the size of the data exposed to the other layers. While our approach preserves model accuracy by avoiding structural changes, this inherently limits the potential speed-up in inference. Model compression techniques, such as pruning and quantization, could further reduce execution time but may require more fine-tuning training to maintain similar accuracy. The combination of the proposed execution pipeline and deployment strategy with model optimization is part of future work, along with parallelization in a distributed manner across the edge-fog-cloud continuum.

V. CONCLUSION

This study defined real-time execution requirements for a deep-learning visual odometry model and proposed algorithmic optimizations to meet these constraints without altering the model structure. The evaluation across Jetson, laptop, and desktop configurations showed that while optimizations significantly reduced execution time (up to 95% improvement on laptop configuration). Jetson hardware still cannot achieve the 30 Hz real-time threshold, making full on-board processing impractical for mobile robots. These findings highlight trade-offs among performance, energy consumption, and hardware feasibility and suggest that distributed processing across edge, fog, and cloud layers may be necessary. Future work will address communication latency, connectivity resilience, and fallback strategies to ensure robust real-time localization in autonomous robotic systems.

REFERENCES

[1] T. Hachijo, S. Igarashi, T. Tani, and M. Indo, "Development of autonomous robots for construction," in *Proc. Int. Symp. Autom. Robot. Construction (IAARC)*, Jun. 2024, pp. 199–205.

[2] Y. Li, J. Park, G. Manogharan, F. Ju, and I. Kovalenko, "A mobile additive manufacturing robot framework for smart manufacturing systems," in *Proc. Manuf. Equip. Automat., Manuf. Processes, Manuf. Syst., Nano/Micro/Meso Manuf., Quality Rel.*, vol. 2, pp. 1–7, Aug. 2024, doi: 10.1115/msec2024-125413.

[3] H. Yin, Q. Sun, X. Ren, J. Guo, Y. Yang, Y. Wei, B. Huang, X. Chai, and M. Zhong, "Development, integration, and field evaluation of an autonomous citrus-harvesting robot," *J. Field Robot.*, vol. 40, no. 6, pp. 1363–1387, Sep. 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/rob.22178

[4] K. U. Singh, A. K. Rai, A. Kumar, S. Kumar, T. Singh, and L. Raja, "Artificial intelligent techniques for disease detection in tomato," in *Proc. IEEE Int. Conf. Contemp. Comput. Commun. (InC4)*, Mar. 2024, pp. 1–6.

[5] G. Bakshi, A. Kumar, and A. N. Puranik, "Adoption of robotics technology in healthcare sector," in *Proc. ICCDN*. Cham, Switzerland: Springer, 2021, pp. 405–414. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-16-2911-2_42

[6] R. Anshul and A. Mohan, "Miniaturized electronics: Advantages, disadvantages and its applications," in *Proc. Nat. Seminar Emerg. Trends Technol. Current Era*, 2022, pp. 28–33.

[7] *Valtronicles | Miniaturization Advantages and Challenges–Valtronic*, Valtronic, Boyertown, PA, USA, 2024.

[8] P. P. Ray, "Internet of Robotic Things: Concept, technologies, and challenges," *IEEE Access*, vol. 4, pp. 9489–9500, 2016.

[9] M. Saravanan, S. K. Perepu, and A. Sharma, "Exploring collective behavior of Internet of Robotic Things for indoor plant health monitoring," in *Proc. IEEE Int. Conf. Internet Things Intell. Syst. (IOTAIS)*, Nov. 2018, pp. 148–154.

[10] A. S. Seisa, G. Damigos, S. G. Satpute, A. Koval, and G. Nikolakopoulos, "Edge computing architectures for enabling the realisation of the next generation robotic systems," in *Proc. 30th Medit. Conf. Control Autom. (MED)*, Jun. 2022, pp. 487–493.

[11] C.-C. Chen, Y.-H. Guan, N. R. Novianda, C.-C. Teng, and M.-H. Yen, "Real-time traffic sign detection for self-driving and energy-saving driving based on YOLOv4 neural network," in *Proc. IEEE/ACIS 23rd Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, Dec. 2022, pp. 208–211.

[12] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robot. Auto. Syst.*, vol. 61, no. 12, pp. 1726–1743, Dec. 2013.

[13] X. Di and R. Shi, "A survey on autonomous vehicle control in the era of mixed-autonomy: From physics-based to AI-guided driving policy learning," *Transp. Res. C, Emerg. Technol.*, vol. 125, Apr. 2021, Art. no. 103008.

[14] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Learn. Knowl. Extraction*, vol. 5, no. 4, pp. 1680–1716, Nov. 2023.

[15] S. Chuprov, E. Marinenkov, I. Viksnin, L. Reznik, and I. Khokhlov, "Image processing in autonomous vehicle model positioning and movement control," in *Proc. IEEE 6th World Forum Internet Things (WF-IoT)*, Jun. 2020, pp. 1–6.

[16] T. Vu, D.-T. Nguyen, D.-T. Nguyen, T. Vu Toan, and V.-T. Tran, "Real-time detection to avoid accidents of interaction between humans and collaborative robot using image processing and machine learning," in *Proc. IEEE 11th Int. Conf. Comput. Cybern. Cyber-Medical Syst. (ICCC)*, Apr. 2024, pp. 000229–000232.

[17] L.-B. Chen, X.-R. Huang, and W.-H. Chen, "Design and implementation of an artificial intelligence of things-based autonomous mobile robot system for pitaya harvesting," *IEEE Sensors J.*, vol. 23, no. 12, pp. 13220–13235, Jun. 2023.

[18] S. Pareigis, T. Tiedemann, N. Schönherr, D. Mihajlov, E. Denecke, J. Tran, S. Koch, A. Abdelkarim, and M. Mang, "Artificial intelligence in autonomous systems. A collection of projects in six problem classes," in *Real-Time Meeting of the Gesellschaft Für Informatik* (Lecture Notes in Networks and Systems), vol. 674. Cham, Switzerland: Springer, 2023, pp. 123–145. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-32700-1_14

[19] S. Wang, R. Clark, H. Wen, and N. Trigoni, "DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 2043–2050, doi: 10.1109/ICRA.2017.7989236.

[20] S.-H. Bach, P.-B. Khoi, and S.-Y. Yi, "Global UWB system: A high-accuracy mobile robot localization system with tightly coupled integration," *IEEE Internet Things J.*, vol. 11, no. 9, pp. 16618–16626, May 2024.

[21] M. Nimura, K. Kanai, and J. Katto, "Accuracy evaluations of real-time LiDAR-based indoor localization system," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2023, pp. 1–5.

[22] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robot.*, vol. 7, no. 66, pp. 1–12, May 2022. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abm6074

[23] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators* (Advanced Textbooks in Control and Signal Processing). London, U.K.: Springer, 2000. [Online]. Available: http://link.springer.com/10.1007/978-1-4471-0449-0

[24] S. MohandSaidi and R. Mellah, "Real-time speed control of a mobile robot using PID controller," in *Proc. Int. Conf. Artif. Intell. Appl.* Cham, Switzerland: Springer, 2022, pp. 548–556. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-96311-8_51

[25] Z. Chen, C. Jiang, and Y. Guo, "Pedestrian-robot interaction experiments in an exit corridor," in *Proc. 15th Int. Conf. Ubiquitous Robots (UR)*, Jun. 2018, pp. 29–34.

[26] S. Zhang, J. Zhang, and D. Tao, "Information-theoretic odometry learning," *Int. J. Comput. Vis.*, vol. 130, no. 11, pp. 2553–2570, Nov. 2022, doi: 10.1007/s11263-022-01659-9.

[27] A. Barodi, A. Bajit, A. Zemmouri, M. Benbrahim, and A. Tamtaoui, "Improved deep learning performance for real-time traffic sign detection and recognition applicable to intelligent transportation systems," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 5, pp. 712–723, 2022. [Online]. Available: www.ijacsa.thesai.org

[28] A. Avramovic, D. Sluga, D. Tabernik, D. Skocaj, V. Stojnic, and N. Ilc, "Neural-network-based traffic sign detection and recognition in high-definition images using region focusing and parallelization," *IEEE Access*, vol. 8, pp. 189855–189868, 2020.

[29] R. Barbosa, O. D. Ogobuchi, O. O. Joy, M. Saadi, R. L. Rosa, S. A. Otaibi, and D. Z. Rodríguez, "IoT based real-time traffic monitoring system using images sensors by sparse deep learning algorithm," *Comput. Commun.*, vol. 210, pp. 321–330, Oct. 2023.

[30] M. A. Munim and M. S. Rahman Kohinoor, "Performance evaluation of deep learning-based facial recognition models on mobile computing environments," in *Proc. IEEE 11th Region 10 Humanitarian Technol. Conf.*, Oct. 2023, pp. 13–18.

[31] T. Alladi, V. Kohli, V. Chamola, F. R. Yu, and M. Guizani, "Artificial intelligence (AI)-empowered intrusion detection architecture for the Internet of Vehicles," *IEEE Wireless Commun.*, vol. 28, no. 3, pp. 144–149, Jun. 2021.

[32] M. Shah, K. Gandhi, B. M. Pandhi, P. Padhiyar, and S. Degadwala, "Computer vision & deep learning based realtime and pre-recorded human pose estimation," in *Proc. 2nd Int. Conf. Appl. Artif. Intell. Comput. (ICAAIC)*, May 2023, pp. 313–319.

[33] M. R. Kounte, J. Rishitha, S. S. Setty, and S. Shivani, "Implementation of realtime design of crowd enumeration via tracking using AI system," in *Proc. Int. Conf. Intell. Innov. Technol. Comput., Electr. Electron. (IITCEE)*, Jan. 2023, pp. 270–274.

[34] T. Natali, L. M. Kurucz, M. Fusaglia, L. S. Mertens, T. J. Ruers, P. J. van Leeuwen, and B. Dashtbozorg, "Automatic real-time prostate detection in transabdominal ultrasound images," *Eur. J. Radiol.*, vol. 191, Oct. 2025, Art. no. 112274. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0720048X25003602

[35] N. Pudasaini, M. A. Hanif, and M. Shafique, "SPAQ-DL-SLAM: Towards optimizing deep learning-based SLAM for resource-constrained embedded platforms," 2024, *arXiv:2409.14515*.

[36] S. Paniego, N. Paliwal, and J. Cañas, "Model optimization in deep learning based robot control for autonomous driving," *IEEE Robot. Autom. Lett.*, vol. 9, no. 1, pp. 715–722, Jan. 2024.

[37] K. S. Chinmay and G. K. Nisha, "Review of autonomous mobile robots," in *Proc. Int. Conf. E-Mobility, Power Control Smart Syst., Futuristic Technol. Sustain. Solutions*, 2024, pp. 1–16.

[38] M. Bültemann, N. Rzepka, D. Junger, K. Simbeck, and H. G. Müller, "Energy consumption of AI in education: A case study," in *Proc. Fachtagung Bildungstechnologien*, in Lect. Notes Informat, 2023, pp. 219–224.

[39] R. Caspart, S. Ziegler, A. Weyrauch, H. Obermaier, S. Raffeiner, L. P. Schuhmacher, J. M. Scholtyssek, D. Trofimova, M. Nolden, I. Reinartz, F. Isensee, M. Götz, and C. Debus, "Precise energy consumption measurements of heterogeneous artificial intelligence workloads," in *Proc. High Perform. Comput., ISC High Perform. Int. Workshops*, in Lecture Notes in Computer Science, 2022, pp. 108–121.

[40] B. Kim, S. Lee, A. R. Trivedi, and W. J. Song, "Energy-efficient acceleration of deep neural networks on realtime-constrained embedded edge devices," *IEEE Access*, vol. 8, pp. 216259–216270, 2020.

[41] G. Cassales, H. M. Gomes, A. Bifet, B. Pfahringer, and H. Senger, "Balancing performance and energy consumption of bagging ensembles for the classification of data streams in edge computing," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 3, pp. 3038–3054, Sep. 2023.

[42] A. Josyula, B. Anand, and P. Rajalakshmi, "Fast object segmentation pipeline for point clouds using robot operating system," in *Proc. IEEE 5th World Forum Internet Things (WF-IoT)*, Apr. 2019, pp. 915–919.

[43] H. Li, J. K. Ng, and T. Abdelzaher, "Enabling real-time AI inference on mobile devices via GPU-CPU collaborative execution," in *Proc. IEEE 28th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2022, pp. 195–204.

[44] W. Niu, Z. Kong, G. Yuan, W. Jiang, J. Guan, C. Ding, P. Zhao, S. Liu, B. Ren, and Y. Wang, "Real-time execution of large-scale language models on mobile," 2020, *arXiv:2009.06823*.

[45] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet 2.0: Evolution of optical flow estimation with deep networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1647–1655.

[46] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734. [Online]. Available: https://aclanthology.org/D14-1179

[47] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[48] R. C. Mello, M. R. N. Ribeiro, and A. Frizera-Neto, "Introduction to cloud robotics," in *Implementing Cloud Robotics for Practical Applications: From Human-Robot Interaction To Autonomous Navigation* (Springer Tracts in Advanced Robotics), vol. 152. Springer, Oct. 2022, pp. 1–11. [Online]. Available: https://link.springer.com/chapter/10.1007/

[49] V. C. Pujol and S. Dustdar, "Fog robotics—Understanding the research challenges," *IEEE Internet Comput.*, vol. 25, no. 5, pp. 10–17, Sep. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9565454/

[50] M. Fallon, H. Johannsson, M. Kaess, and J. J. Leonard, "The MIT stata center dataset," *Int. J. Robot. Res.*, vol. 32, no. 14, pp. 1695–1699, Dec. 2013. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364913509035

[51] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3354–3361.

[52] (2025). *Executors—ROS 2 Documentation: Humble Documentation*. [Online]. Available: https://docs.ros.org/en/humble/Concepts/Intermediate/About-Executors.html

[53] A. Isenko. (2023). *Basic Hardware Monitor*. [Online]. Available: https://github.com/cirquit/py-hardware-monitor

[54] R. Bonghi. (2007). *Jetson-Stats*. [Online]. Available: https://github.com/rbonghi/jetson_stats

[55] (2024). *Jetson AGX Orin 32 GB Vs GTX 1660*. [Online]. Available: https://technical.city/en/video/GeForce-GTX-1660-vs-Jetson-AGX-Orin-32-GB

[56] C. Melançon, G. Simard, M. Saad, K. Kaur, and J. Gascon-Samson, "BlazeFlow: A multi-layer communication middleware for real-time distributed IoT applications," in *Proc. 1st Int. Workshop Middleware Comput. Continuum*, Dec. 2023, pp. 30–35.

CÉDRIC MELANÇON (Member, IEEE) received the B.Eng. degree in electrical engineering and the M.Eng. degree in aerospace engineering from École de technologie supérieure, Montreal, in 2003 and 2005, respectively, and the Ph.D. degree from the Electrical Engineering Program, École de technologie supérieure, in 2020. He is currently a Ph.D. Candidate with the Electrical Engineering Program, École de technologie supérieure. He has approximately 18 years of experience in various industries, including embedded design, simulation, automation, distributed systems, and the Internet of Things. His current research aims to develop an AIoT platform that enables the distributed control of autonomous robots to assist nurses in their work.

MAAROUF SAAD (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from École Polytechnique of Montreal, Montreal, QC, Canada, in 1982 and 1984, respectively, and the Ph.D. degree in electrical engineering from McGill University, Montreal, in 1988. In 1987, he joined École de technologie supérieure, Montreal, where he teaches control theory and robotics courses. His research interests include nonlinear control and optimization applied to robotics.

KULJEET KAUR (Member, IEEE) is currently an Associate Professor with École de technologie supérieure (ÉTS), Montreal. Before this, she was an NSERC Postdoctoral Fellow with ÉTS and a Visiting Researcher with Nanyang Technological University, Singapore. She has secured more than 65+ research articles in top-tier journals and international conferences. Her main research interests include cloud computing, information security, energy efficiency, smart grid, frequency support, and vehicle-to-grid. She received the 2018 IEEE ICC Best Paper Award, the 2019 Best Research Paper Award from Thapar Institute of Engineering and Technology, India, and the 2020 IEEE System Journal Best Paper Award.

JULIEN GASCON-SAMSON (Member, IEEE) received the B.Eng. and M.Eng. degrees in software and computer engineering from Polytechnique Montréal, and the Ph.D. degree in computer science from McGill University. He is currently a Professor with the Software and IT Engineering Department, École de technologie supérieure, (ÉTS), Montréal. He holds the ÉTS Marcelle-Gauvreau Engineering Research Chair on Applications and Services in edge computing. He has led several projects with significant impacts in the areas of edge computing and publish-subscribe systems, which led to several publications in high-profile system venues. His research interests are in systems-related areas, and include distributed systems, publish/subscribe communication paradigms, cloud/edge computing, and distributed stream processing.