Resource-Efficient Sensor Fusion at the Edge via System-Wide Dynamic Gated Neural Networks

Chetna Singhal, Yashuo Wu, Francesco Malandrino, Sharon G L Contreras, Marco Levorato, Carla Fabiana Chiasserini

Resource-Efficient Sensor Fusion at the Edge via System-wide Dynamic Gated Neural Networks

Chetna Singhal, *Senior Member, IEEE,* Yashuo Wu, Francesco Malandrino, *Senior Member, IEEE,*
Sharon G. L. Contreras, Marco Levorato, *Senior Member, IEEE,* Carla Fabiana Chiasserini, *Fellow, IEEE*

***Abstract*—Next-generation mobile systems will support multiple AI-based applications, each leveraging heterogeneous sensors and data sources through deep neural network (DNN) architectures collaboratively executed within the network. In this context, to minimize the cost of the AI inference task subject to requirements on latency, quality, and – crucially – *reliability* of the inference process, it is vital to optimize (i) the set of sensors/data sources and (ii) the DNN architecture, (iii) the network nodes executing sections of the DNN, and (iv) the resources to use. To achieve these goals, we leverage dynamic gated neural networks with branches, and propose a novel algorithmic strategy called Quantile-constrained Inference (QIC), based upon quantile-Constrained policy optimization. QIC makes joint, high-quality, swift decisions on all the above aspects of the system, with the aim to minimize inference energy cost. We remark that this is the first contribution connecting gated dynamic DNNs with infrastructure-level decision making. We evaluate QIC using a dynamic gated DNN with stems and branches for optimal sensor fusion and inference, trained on the RADIATE dataset offering Radar, LiDAR, and Camera data, and real-world wireless measurements. Our results confirm that QIC closely matches the optimum and outperforms existing approaches in reducing energy consumption (compute, communication, and total) and application requirements failure by over 70%.**

***Index Terms*—Network support to machine learning, Dynamic deep neural networks, Energy efficiency, Inference in the mobile-edge continuum**

I. Introduction

Several emerging mobile applications are powered by artificial intelligence (AI) algorithms, often processing and fusing the data produced by multiple sensors and *data sources* [1], [2]. Many of such algorithms take the form of deep neural networks (DNNs). An intriguing class of neural architectures include *branches*, also known as early exits [3]. These architectures are dynamic, as the execution adapts to the characteristics of the input: the branches are sequentially executed until a satisfactory output is produced. The state of the art in neural networks evolved past early exits, and recently models with more structured and complex adaptability were developed. Specifically, these models are articulated into sections, connected by neural structures called *gates*. The gates directly control the internal routing of information based on optimal execution strategies learned during training [4]. In their simplest form, gates pre-select a full model, or a set of models in mixture of experts settings, based on simple features of the input. In more complex instances, such as that in [2] and the one considered in this paper, gated models are crafted as the composition of stems extracting features from a diverse set of sensors, and branches that process these intermediate features to produce a final output. The gates, then, control the activation of the stems, and the way features referring to different input sensors are fused and analyzed, and, in the model developed in this paper, the modality of sensor fusion. This class of models, whose design and training are highly non-trivial, perform a dynamic and context-aware form of sensor fusion.

Current literature develops and analyzes these models in isolation, and considering execution on a single device [2]. In contrast, **this paper represents the first contribution considering dynamic gated models in the context of layered computing/communication and memory infrastructures – i.e., those composed of interconnected mobile nodes and edge servers.** In such setting, these architectures represent a significant opportunity, as they enable a flexible and efficient allocation of computing, communication, and memory load across different layers of the system. Importantly, as the gates control the structure of the neural network model and the use of data sources (that is, the combination of *stems* and *branches* mapping the input to the output) in response to input characteristics, the whole resource allocation strategy becomes **context and input aware**. We, thus, define the gates as connected to an infrastructure-level orchestrator that directly controls the activation of the DNN sections, and the resources on which they will be executed. That is, the model can be split at the gates and executed on different system resources (mobile nodes and edge server). Such characteristics and interplay between the inner neural network structure and the operations of the infrastructure becomes especially important when considering multiple applications – each with different inference quality and latency requirements – coexisting on the same resources.

The use of such architectures therefore results in (potentially) effective data analysis, improved efficiency, and lower costs. At the same time, achieving these results requires swift, high-quality, and *joint* decisions about such diverse aspects as:

1) the data sources to leverage for each application;
2) the DNN sections (i.e., stems and branches) to use;
3) the network nodes where stems and branches shall run;

C. Singhal is with ERMINE, Inria centre at University of Rennes, France. Y. Wu, S. G. L. Contreras and M. Levorato are with University of California, Irvine, USA. F. Malandrino and C. F. Chiasserini are with CNR-IEIIT and CNIT, Italy. C. F. Chiasserini is with Politecnico di Torino, Italy, and Chalmers University of Technology, Sweden.

This work was supported through SNS JU under the EU's HE research and innovation programme under Grant Agreement No. 101192521. Part of the work was also supported by the US NSF under grant CCF 2140154.

4) the computation, communication, and memory resources to devote to each application and at each node.

All these decisions have to be made with the goal of minimizing the cost (e.g., energy) of inference, subject to inference quality (e.g., mean absolute precision, mAP) and latency requirements. We remark that each data source is connected to a different *stem* of the DNN, which produces features that are then processed by a *branch* tailored to the quantity and type of the data the source produces (e.g., 2D or 3D images), to generate the final inference output. Also, importantly, we do not express inference quality requirements in terms of average/expected values (e.g., the expected mAP), but rather in terms of a target *quantile* thereof (e.g., the 90th percentile of mAP). On the one hand, this reflects the time- and performance-critical nature of many modern applications, which need to make correct decisions with a *guaranteed* (high) probability. At the same time, the added flexibility of targeting arbitrary quantiles also accommodates the opposite scenario: applications for which occasional failures are acceptable and/or have limited consequences can trade some inference quality for a lower cost.

On the negative side, making all the decisions mentioned above while accounting for inference quality quantiles is dauntingly complex, for three main reasons. The most straightforward is the *scale* of the problem, e.g., the very large number of alternatives to consider. Furthermore, the combinatorial *structure* of the problem (coming from, among other things, the presence of a discrete set of stems and branches to choose from) rules out direct optimization approaches. Finally, and most important, the *nature* of the problem itself is utterly new, which makes existing approaches, like those discussed in Sec. II, impossible to apply to our scenario.

In this work, we fill this gap by proposing a new solution strategy called *Quantile-constrained Inference based on quantile-Constrained policy optimization* (QIC). QIC makes decisions that are: (i) *joint*, as they account for data sources, stems, branches, network nodes and resources to use for each application; (ii) *dependable*, as they can guarantee an arbitrary value of an arbitrary quantile of the inference quality; (iii) *efficient and effective*, as near-optimal choices are made in polynomial time. Our contributions in this work go beyond QIC itself and can be summarized as follows:

• We provide (Sec. III) a comprehensive description of the applications and scenario we target, along with (Sec. IV) a synthetic, yet expressive, model accounting for all the most relevant features of the system;

• We develop (Sec. III-B) a new instance of dynamic gated neural model for object detection performing sensor fusion on Camera, Light Detection and Ranging (LiDAR), and Radio Detection And Ranging (Radar) data. Notably, the model's internal configuration is connected to infrastructure-level decisions, and its sections can be deployed over different system's resources;

• We develop the QIC solution framework, which innovatively applies quantile-constrained optimization on the dynamic graph representing the system evolution over time, and characterize its efficiency (Sec. V);

• We build a highly realistic reference scenario, including state-of-the-art DNNs and real-world wireless measurements (Sec. VI);

• In such a scenario, we study the performance of QIC (Sec. VII), finding it to significantly outperform state-of-the-art approaches (70% reduction in both energy and application requirements failure) and closely match the optimum.

Finally, we remark that we have made our solution and evaluation framework publicly available on github (https://github.com/chetna-iitd/Gated_Sensor_fusion).

II. Related Work

Dynamic DNNs have gained popularity for adding flexibility to model inference processes. The concept began with the implementation of early exits for accelerated inference as introduced in [5], [6]. Subsequently, dynamic models evolved to incorporate multi-branch architectures. These models have been pivotal in enhancing both the quality and efficiency of deep learning-based object detection, and image classification [7]. Specifically, [7] adopts a parallel multi-branch architecture to improve object detection accuracy without additional computational overhead. This concept of multi-branch architecture is also explored in [8] where the system maintains a shared knowledge base and common features before diverging into distinct branches. [9] employs multiple expert models in the architecture that are activated by a gating encoder based on the task and input sample. Similar to the mixture of experts, a stem-branch architecture is used for multi-class and multi-label classification to improve accuracy in [10]. In [11], the semantic-based deep network architecture consists of a stem, multiple branches, and a gating mechanism, to improve the computational efficiency of the inference task in image classification. However, **prior work has considered input data from only one source or the fixed deployment on a single computing node. In our work, we explore the use of multiple data sources and the dynamic deployment of the stem-branch architecture across the network nodes in an energy-efficient manner**.

Inference of DNNs using collaborative computation between mobile device and cloud was modeled using a directed acyclic graph in [1]. There, the adaptive model parameters account for the resource availability and the model layers are processed on the mobile device and the cloud server. However, [1] does not consider the dynamic variation of system parameters, multiple input sources, the stem-branch architecture, or the reliability of the inference process. While [2] delves into sensor fusion, it predominantly focuses on the training phase, neglecting the inference stage. On the other hand, [12] investigates energy-efficient inference processes but does so by selecting all available inputs, resulting in suboptimal energy utilization. Energy considerations are instead made in [6], [13], which develops a framework to deploy DNNs with early-exits in distributed networks.

As for graph-based modeling of real applications, the dynamic graph model-based optimization [14] has been used for wireless vehicular networks [15] and device-to-device communication [16]. The work therein proposes a framework for

the split deployment of such a context-based specialized DNN model architecture in distributed network systems. Further, dynamic graph modeling for real-time applications requires multi-constrained temporal path discovery. In [17] and [18], this has been solved using two-pass approximation algorithm, TPA, and adaptive Adaptive Monte Carlo Tree Search algorithm, AMCTS, respectively. In our work, we compare the performance of our proposed framework to these state-of-the-art algorithms.

Finally, we mention that an early version of our work appeared in our conference paper [19]. Therein we presented a simpler problem, we did not give any proof of the problem complexity, and considered only small-case, simplified scenarios for our experimental analysis.

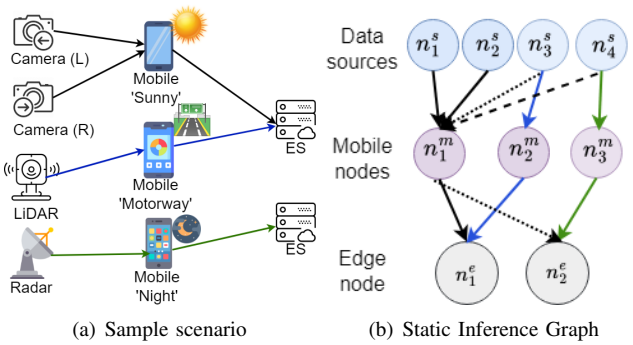
III. System Scenario

This section first introduces the system scenario we consider by characterizing the data sources and the network nodes composing the mobile-edge continuum, as well as their interaction. Then it describes the dynamic gated DNN model that we developed and that we take as reference neural network model for our study.

A. System components

We consider a network system consisting of data sources: camera (Left/Right), Radar, LiDAR, mobile devices, and edge servers. As discussed in detail later in the paper, the notion of *context* is instrumental for the overall optimization, and the DNN configuration is greatly influenced by environmental conditions. Each mobile device is associated with an environmental context (e.g., `Sunny`, `Motorway`, and `Night`), which, as discussed later, drives performance given the configuration. Intelligent applications (denoted in Fig. 1(a) using different arrow colors) require the inference task to be performed with the required level of performance guarantees (latency and mAP). In the example scenario, Application 1 (black arrows) uses Camera (L-R) input on a mobile node in a `Sunny` context and the dynamic DNN for this application is deployed on the mobile node and the edge server (ES). Application 2 (blue arrows) uses LiDAR input and the dynamic DNN is deployed on the mobile node with the `Motorway` context and on the ES. The same ES is used to deploy both Applications 1 and 2. Application 3 (green arrows) uses Radar input and the DNN model is deployed on the mobile node with `Night` context and on a different ES.

The devices acting as data sources, the mobile nodes, and the ESs are connected over a wireless network (i.e., data sources could be either co-located with mobile nodes or connected to them). The communication resources of the network, the memory and the computation resources of the mobile nodes and ESs are shared for executing the inference tasks using the dynamic deployment of a pre-trained dynamic DNN model. To effectively support the applications, an orchestrator, located in the network infrastructure, (i) determines the data sources to be used to feed the DNN for the different applications, and, accordingly, instructs the mobile nodes about the data to be collected, (ii) instructs both the mobile nodes and the edge servers about the sections of the DNN (stems or branches, or both) to deploy locally, and generates the configuration to be used by the gates controlling the data processing throughout the DNN.



(a) Sample scenario (b) Static Inference Graph

Fig. 1. (a) System scenario with 3 applications, 4 data sources, 3 mobile nodes, each associated with a different context, and 2 edge servers. (b) Graph-based model of the system configuration.

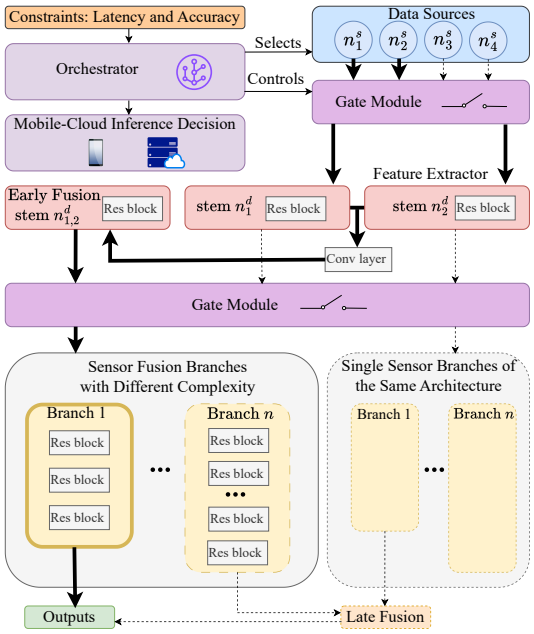


Fig. 2. Architecture implemented for object detection inference, illustrating the process where the orchestrator selects the data source, followed by the gate modules determining the appropriate stem and branch for deployment.

B. Dynamic DNN model

One of the core contributions of this paper is accounting for the deep interplay between innovative dynamic neural models and the resource allocation strategy of the collaborative edge computing system described previously. In this section, we present the instance of dynamic gated neural network model that we developed. At a high level, the architecture performs dynamic, adaptive and context-aware sensor fusion on Camera, LiDAR, and Radar data for object detection. Fig. 2 summarizes the structure of the model. Our model architecture adopts and extends the HydraFusion framework [2], introducing a scalable design to accommodate the varying computational demands of the applications to be supported via our gated mechanism. We remark how the adaptation of the ability of the model to

change the computing workload is instrumental to integrate the models in a system-wide resource allocation framework.

As illustrated in Fig. 2, a gate module controls the activation of the stems, determining how features from different input sensors are fused and analyzed. Contrary to the gating mechanism employed in [2], which selects branches during the training phase to enhance inference quality (mAP), **our study implements the gating mechanism during the inference phase,** in order to optimize the tradeoff between output mAP, energy expense, and latency given the current context.

Our neural model architecture processes input data from various modalities to facilitate object detection. We employ ResNet-18/50/101 [20] as the foundational architecture, integrated within a faster Recursive-Convolutional Neural Network (R-CNN) framework. Specifically, initial sensor data from various modalities are analyzed by distinct CNNs, referred to as "stems". These stems, serving as feature extractors, are specialized to process the respective sensor inputs, transforming them into initial sets of features. These stems correspond to the first block of the ResNet architecture. An early fusion mechanism concatenates the features from each stem, resulting in three augmented stems that enhance context-specific inference quality (mAP). A 2D convolution layer merges these concatenated features, which are inputs to the subsequent ResNet layers, termed "branches".

The architecture consists of single-sensor branches as Left Camera, Right Camera, LiDAR, and Radar, alongside early fusion branches combining Left and Right Cameras, LiDAR and Radar, and Left Camera and LiDAR. This configuration results in six branches, each presenting three levels of complexity aligned with the ResNet 18/50/101 models, resulting in a total of 18 selectable branches. A second gate module determines the most suitable branch or branches for the task.

Furthermore, we integrate late fusion as a post-processing step exclusively for the (less complex) ResNet18 branches, constrained by computational efficiency. We applied Non-Maximum Suppression (NMS) as our late fusion mechanism to the outputs from the ResNet-18 branches to balance deployment efficiency. Indeed, empirical evidence from our pre-trained models indicates that late fusion improves inference quality (mAP). However, it necessitates the complete deployment of multiple branches, which escalates energy consumption. Consequently, our gating mechanism consistently avoids selecting late fusion for ResNet-50 and ResNet-101 branches.

Fig. 2 illustrates the decision process, highlighted by bold arrows, starting from the selection of data sources, leading to the choice of early fusion branches of a specific complexity level, excluding late fusion. Dashed lines represent potential, yet unselected, pathways available to the gating module.

We remark how different configurations of the gates result in a different computing load associated with stems and branches, as well as a different data flow input-to-stems and stems-to-branches. Moreover, this interrelation is heavily dependent on the context. Intuitively, these quantities are quintessential to define resource allocation strategies. We also underline how the model is splittable, meaning that stems and branches can be executed on different nodes. This results in a data flow and computing load between mobile nodes and edge servers that is dependent on the configuration and on the allocation of computing tasks.

IV. System Model and Problem Formulation

We represent the above system scenario by defining a set of heterogeneous network nodes, $\mathcal{N}=\{\mathcal{N}^{\mathrm{s}}\cup\mathcal{N}^{\mathrm{m}}\cup\mathcal{N}^{\mathrm{e}}\}$, composed of the following subsets: 1) data sources (hence, data modes) in $\mathcal{N}^{\mathrm{s}}$; 2) mobile nodes (which can host either only the stems or the stems along with the branches of the DNN model) in $\mathcal{N}^{\mathrm{m}}$; 3) edge servers (ESs) (which can host either only the branches or the stems along with the branches of the DNN model), in $\mathcal{N}^{\mathrm{e}}$. We indicate with $N=|\mathcal{N}|$ the total number of nodes; similarly, N^{s}, N^{m}, N^{e} denote (respectively) the number of sources, mobile nodes, and edge servers.

For each node $n\in\mathcal{N}^{\mathrm{s}}\cup\mathcal{N}^{\mathrm{m}}$, the amount of (uplink) communication resources at its disposal, e.g., the number of resource blocks a mobile node can use to communicate with the edge servers, is denoted with B_n. Similarly, C_n and M_n denote the amount of (resp.) computational (e.g., CPU or GPU) and memory resources (in number of executable instructions/s and Mb, resp.) available at node $n\in\mathcal{N}^{\mathrm{m}}\cup\mathcal{N}^{\mathrm{e}}$, which can be used for sensor fusion and inference.

Applications are denoted by elements $h\in\mathcal{H}$, and are associated with a maximum latency requirement $\ell^h_{\omega,\max}$ and a minimum inference quality (mAP) requirement $\alpha^h_{\omega,\min}$, *both expressed in terms of the quantile* $\omega\in[0,1]$. Considering different quantiles allows us to balance efficiency (e.g., energy consumption) against inference quality and latency guarantees. We can thus limit the extent of an undesirable event (failing to meet the application requirements) by tweaking ω, depending upon the scenario and application at hand. Intuitively, one might use more lightweight models in scenarios/applications where occasional failures can be accepted. On the other hand, critical scenarios where inference quality (mAP) guarantees are required call for more robust models – even if they have more substantial resource requirements. We remark how the resource usage versus inference quality performance tradeoff is informed by the context.

For each application h, we have a set of possible data sources $\mathcal{N}^h\subseteq\mathcal{N}^{\mathrm{d}}$, corresponding to nodes equipped with a sensor (e.g., Camera, LiDAR, or Radar) that can be used for application h. The quantity δ^h_n represents the quantity of data (in bits) emitted by source n when used for application h.

Also, each application is associated with a *splittable* dynamic DNN model, composed of stems $\mathcal{A}^{\mathrm{s},h}$ and branches $\mathcal{A}^{\mathrm{b},h}$ (although simply referred to as branches, the latter ones may come with an additional stem at the end, as described in Sec.III-B). More specifically, the model can be split by deploying stems and branches at different nodes. For each stem and branch in $a\in\mathcal{A}^{\mathrm{s},h}\cup\mathcal{A}^{\mathrm{b},h}$, and for each node $i\in\mathcal{N}^{\mathrm{m}}\cup\mathcal{N}^{\mathrm{e}}$, we know the quantity δ^h_a outgoing from stem (or branch) a when used by application h. For each stem $a\in\mathcal{A}^{,h}$ (branch $a\in\mathcal{A}^{\mathrm{b},h}$) and application h, we are also given the computational complexity of each stem (branch), expressed in number of operations o^h_a (e.g., in CPU cycles per bit of incoming data) necessary to run the stem (branch) as well as the memory required to deploy the stem (or branch), expressed in bits υ^h_a.

TABLE I
NOTATION

Symbol	Definition
$n \in \mathcal{N}$	Heterogeneous network nodes
$h \in \mathcal{H}$	Applications
C_n	Computation resource [in TOPS] of n
B_n	Communication resource blocks at n
M_n	Memory availability [in bits] of n
$\mathcal{A}^{\mathrm{b},h}$	Branches of of application h
$\mathcal{A}^{\mathrm{s},h}$	Stems of of application h
δ_n^h	Data (in bits) emitted by n for application h
o_a^h	Number of operations to run $a \in \mathcal{A}^{\mathrm{s},h} \cup \mathcal{A}^{\mathrm{b},h}$
v_a^h	Memory required to deploy $a \in \mathcal{A}^{\mathrm{s},h} \cup \mathcal{A}^{\mathrm{b},h}$
$\sigma(h,n)$	Decision variable to deploy $a \in \mathcal{A}^{\mathrm{s},h} \cup \mathcal{A}^{\mathrm{b},h}$ on n
ϵ_n^c	Energy cons. of a computation resource at n
ϵ_n^b	Energy cons. of a communication resource at n
$\epsilon(\boldsymbol{\sigma}, b, c)$	Overall system energy consumption
c_n^h	Compute resource [instructions/s] at n for h
m_n^h	Memory resource [bits] at n for h
b_n^h	Commun. resource [resource blocks)] at n for h
$\alpha_\omega^h(\boldsymbol{\sigma})$	ω quantile of inference quality (mAP)
$\ell_\omega^h(\boldsymbol{\sigma}, b, c)$	ω-quantile of the sensor fusion and inference time

Example: Static sensor fusion and inference graph. Fig. 1(b) is a static sensor fusion and inference graph, exemplifying the system nodes and the configuration used to support the three different applications shown in Fig. 1(a). It depicts four data sources in $\mathcal{N}^{\mathrm{s}}$ (blue circles), three mobile nodes in $\mathcal{N}^{\mathrm{m}}$ (pink circles), and two edge servers in $\mathcal{N}^{\mathrm{e}}$ (grey circles). Arrows of different colors correspond to different applications in $\mathcal{H}$ and solid arrows between nodes indicate the deployment decision that causes the flow of information between network nodes; as an example, the blue application uses data source n_3^{d} to get data, mobile node n_2^{m} to deploy its stem, and edge server n_1^{e} for its branch. Notice that multiple data sources can be used as needed (as in the case of the black application) as well as multiple stems and branches. We have indicated alternative deployment solutions for the black application using dotted and dashed black arrows in the graph.

Problem formulation. Generalizing from the example above, the decisions the orchestrator needs to make are: (1) the data source(s), stem(s), and branch(es) to use for each application; (2) where to deploy them; (3) how to distribute the available computation, memory, and communication resources available at nodes across applications. Once it makes the decisions (1) and (2), then the orchestrator also generates the logic to be executed by the gates of the dynamic neural model. We express the first two decisions through variables $\sigma(h,n) \in \mathcal{A}^{\mathrm{s},h} \cup \mathcal{A}^{\mathrm{b},h} \cup \{\mathsf{data}, \emptyset\}$, expressing how application $h \in \mathcal{H}$ uses node $n \in \mathcal{N}$. Such variables can take the following values:

- $\emptyset$, if node n is not used by application h;
- data, if that node is a data source used by application h;
- a value in $\mathcal{A}^{\mathrm{s},h}$ or $\mathcal{A}^{\mathrm{b},h}$, if application h uses node n to deploy a stem or branch (respectively).

We also indicate with $\boldsymbol{\sigma}$ (in bold) the $|\mathcal{H}| \times |\mathcal{N}|$ matrix collecting the values of all σ-variables. The notation is summarized in Table I.

Concerning resource allocation, we indicate with c_n^h (in no. instructions/s), m_n^h (in no. of bits), ad b_n^h (in no. of resource blocks), respectively, the computational, memory, and radio resources that are allocated to application h at node n, and with ρ_n the per-resource block bit rate associated with the highest Modulation and Coding Scheme (MCS) that node n can use for uplink transmissions. Accordingly, R_n^h denotes the (outgoing) data rate available to application h at node n, i.e., $R_n^h = b_n^h \cdot \rho_n$.

Given the values of the above decision variables, the system performance can be derived as follows. The overall system energy consumption is driven by the computational and communication resources allocated at each node, hence:

$$\epsilon(\boldsymbol{\sigma}, b, c) = \sum_{n \in \mathcal{N}} \sum_{h \in \mathcal{H}} \left(\epsilon_n^c c_n^h + \epsilon_n^b b_n^h \right), \tag{1}$$

where ϵ_n^c and ϵ_n^b represent the energy consumption associated with the usage of each unit of (resp.) computational and communication resources at node n.

Concerning the time for sensor fusion and inference, it includes two components, namely, the computational latency and the network latency. Given application h, the former is:

$$\ell_{\mathrm{c}}^h(\boldsymbol{\sigma}, c) = \sum_{n \in \mathcal{N} :\ \sigma(h,n) \in \mathcal{A}^{\mathrm{s},h} \cup \mathcal{A}^{\mathrm{b},h}} \frac{o_{\sigma(h,n)}^h}{c_n^h},$$

while the latter is given by:

$$\ell_{\mathrm{b}}^h(\boldsymbol{\sigma}, b) = \sum_{n \in \mathcal{N} :\ \sigma(h,n) = \mathsf{data}} \frac{\delta_n^h}{R_n^h} + \sum_{n \in \mathcal{N} :\ \sigma(h,n) \in \mathcal{A}^{\mathrm{s},h} \cup \mathcal{A}^{\mathrm{b},h}} \frac{\delta_{\sigma(h,n)}^h}{R_n^h}.$$

In other words, computational latency is incurred for each node that is used for either a stem or a branch (top), while network latency (bottom) accounts for the size of data and available data rate. Combining the above, the total latency for application h is:

$$\ell^h(\boldsymbol{\sigma}, b, c) = \ell_{\mathrm{c}}^h(\boldsymbol{\sigma}, c) + \ell_{\mathrm{b}}^h(\boldsymbol{\sigma}, b), \tag{2}$$

and we indicate with $\ell_\omega^h(\boldsymbol{\sigma}, b, c)$ the ω-quantile of the sensor fusion and inference time and with $\omega \in [0,1]$ the target *quantile* we are interested into. Notice that, in general, $\ell_\omega^h(\boldsymbol{\sigma}, b, c)$ depends upon the distribution of the bitrate resulting from the selected MCS. Concerning inference quality (mAP), its distribution and quantiles can be estimated through several methodologies [2], [21], [22], [23], yielding the inference quality *quantile* $\alpha_\omega^h(\boldsymbol{\sigma})$.

Considering a **snapshot of the system under study modeled through the static sensor fusion and inference graph**, and combining all the above, we can formulate the orchestra-

tor's decision problem as the following optimization problem:

$$\min_{\boldsymbol{\sigma},b,c,m} \ \epsilon(\boldsymbol{\sigma},b,c) \tag{3a}$$

$$\text{s.t. } \alpha^h_\omega(\boldsymbol{\sigma}) \geq \alpha^h_{\omega,\min} \qquad \forall h \in \mathcal{H} \tag{3b}$$

$$\ell^h_\omega(\boldsymbol{\sigma},b,c) \leq \ell^h_{\omega,\max} \qquad \forall h \in \mathcal{H} \tag{3c}$$

$$m^h_n \geq \sum_{a=\sigma(h,n)} v^h_a \qquad \forall h \in \mathcal{H}, \forall n \in \mathcal{N} \tag{3d}$$

$$\sum_{h\in\mathcal{H}} c^h_n \leq C_n \qquad \forall n \in \mathcal{N} \tag{3e}$$

$$\sum_{h\in\mathcal{H}} m^h_n \leq M_n \qquad \forall n \in \mathcal{N} \tag{3f}$$

$$\sum_{h\in\mathcal{H}} b^h_n \leq B_n \qquad \forall n \in \mathcal{N}. \tag{3g}$$

The orchestrator seeks to minimize the total energy consumption (3a), subject to the constraints that all applications achieve their target inference quality quantile (3b) within their target application latency (3c). In doing so, the orchestrator must be mindful of the total amount of computational (3e), memory (3f), and radio (3g) resources available at each node. The memory allocation at a node for an application must meet the memory requirement for the stem or the branch deployed on that node (3d). Intuitively, obtaining a better performance (thus satisfying (3b) and (3c)) tends to require more resources, but doing so would increase the energy consumption (3a). At the same time, (3e), (3f), and (3g) prevent the orchestrator from using particularly desirable (e.g., well-connected) nodes beyond their capabilities.

Problem complexity. Even the *static* version of our problem, i.e., minimizing (3a) subject to constraints (3b)–(3g), is extremely complex, hence, impractical to solve directly for large instances.

Property 1: The problem of minimizing (3a) subject to constraints (3b)–(3g) is NP-hard.

Proof: We use a reduction from the knapsack problem, which is known to be NP-hard [24]. We consider a set $\mathcal{I}=\{i\}$ of items, each with weight w_i and value v_i, along with a maximum weight $w^{\max}$. We have to decide whether to take each item i with the goal of maximizing the total value, subject to the constraint that the total weight does not exceed $w^{\max}$. We then map any instance of the knapsack problem into a – heavily simplified – instance of ours, where: (i) there is only one application and one stem and the inference quality requirement is $\alpha^h(\boldsymbol{\sigma})_{\min}=0$; (ii) there are $|\mathcal{I}|$ data sources and exactly one possible branch for each data source; (iii) there is only one physical node with infinite capabilities (and, hence, network transfers are infinitely fast); (iv) computing times are given. At this point, our only decision is whether or not to use each data source (hence, also activate the branch using it); importantly, selecting a data source corresponds to *not* selecting the corresponding item in the knapsack problem.

Specifically, we start from a state where all data sources (hence, all branches) are used, which corresponds to taking no items in the knapsack problem. In this situation, the inference quality is $\alpha^h(\boldsymbol{\sigma})=w^{\max}$. Then, we set the inference quality and energy consumption functions of our problem in such a way that removing the data source (and the branch) corresponding to item i in the knapsack problem (i) improves the objective (3a) (i.e., reduces the energy consumption) by v_i, while (ii) also reducing the inference quality (3b) by w_i. The optimal solution to our problem will have the lowest possible energy consumption compatible with keeping the inference quality above zero. This is also the optimal solution to the knapsack problem, which requires to maximize the value without exceeding the weight limit. Hence, the reduction is complete. It can be seen by inspection that such a reduction is polynomial in complexity – indeed, constant, as it has no loops. Thus, the thesis is proved. ■

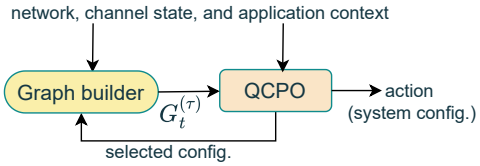


Fig. 3. QIC solution framework.

In summary, although helpful to formalize the problem that the orchestrator needs to face, *the problem complexity and the fact that the mobile network system and context are dynamic in nature* (they both vary with time) demand for an efficient algorithmic solution strategy. Below, we address this need by proposing our QIC solution, which effectively and efficiently copes with both the system complexity and dynamics.

V. QIC: A Dynamic Dependable Solution Framework

Given the dynamic nature of the system, in our solution approach we introduce the notion of time by extending the static sensor fusion and inference graph model (discussed in the previous section) to an *attributed dynamic graph model*. Then, in light of the problem complexity, we also apply the *Quantile Constrained Policy Optimization (QCPO) reinforcement learning algorithm [25] to the attributed dynamic graph and find the efficient set of data sources to be used and the DNN deployment configuration and resource allocation* in the dynamic system for heterogeneous applications.

A schematic illustration of our proposed QIC framework is presented in Fig. 3. It consists of two blocks: *the Graph Builder* and the *QCPO*. Given time $t=0$, the first block creates the initial attributed dynamic graph, $G_t^{(0)}$, i.e., the attributed graph reflecting the system at the initial time instant. The QCPO block instead performs quantile constrained reinforcement learning (RL) [25]. It takes $G_t^{(0)}$ as input and selects the action, i.e., the configuration (DNN stems and branches, where they are deployed and the corresponding resource allocation), that maximizes a reward function matching the objective in (3a). Based on the selected action, the Graph Builder updates the attributed dynamic graph, yielding $G_t^{(1)}$. The procedure is repeated for $P_{\max}$ epochs, thus generating $G_t^{(\tau)}$ at every epoch τ, and the solution to be enacted at time t will be given by the action selected in the last epoch.

Below, we give further details on our solution approach.

A. Attributed dynamic graph model

We account for the temporal variations of the active applications, network conditions, and context (e.g., `Sunny`, `Motorway`, `Night`) by combining the static sensor fusion and inference graph representation of the system (Fig. 1) given for each time instant into an attributed dynamic graph model (Fig. 4). By doing so, the attributed dynamic graph can represent a time-based deployment of dynamic DNN models with stems and branches in the mobile-edge continuum to provision sensor fusion and inference tasks for heterogeneous applications.

Each edge in an attributed dynamic graph contains multiple dynamic attributes, each specifying a different system-level constraint. This facilitates the identification through the graph of the solution to our time-varying, multi-constrained problem.

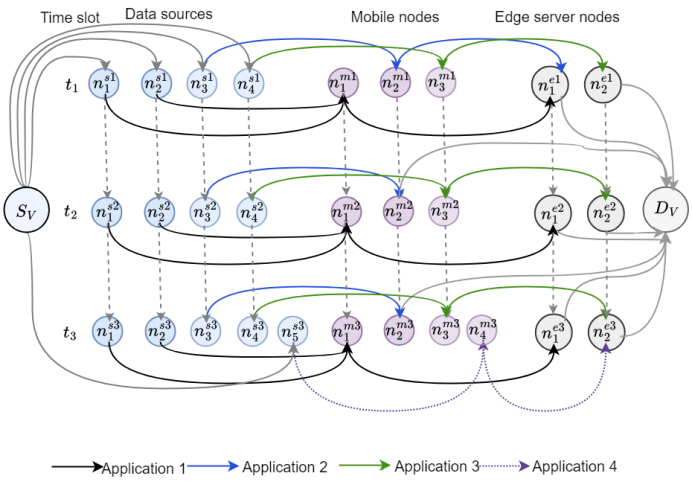


Fig. 4. Dynamic sensor fusion and inference graph over three time slots. At t_1, it reflects the example in Fig. 1; at t_3, it includes 4 applications, 5 data sources, 4 mobile nodes, and 2 edge servers.

The dynamic graph is denoted with $\mathcal{G}=\{G_1, G_2, \ldots G_T\}$, where:

- $G_t=\{V_t, E_t, \mathcal{F}_t\}$, $t=1,\ldots T$, is the graph representing the system at time t;
- V_t is the set of vertices of G_t, representing data sources $n_i^{st}, i \in \{1,2,\ldots N^s\}$, mobile nodes $n_i^{mt}, i \in \{1,2,\ldots N^m\}$, and edge servers n_i^{et}, $i\in\{1,2,\ldots N^e\}$, plus a source, S_v, and a destination, D_v, as fictitious vertices representing (resp.) the starting and ending point of the system configuration process;
- E_t is the set of edges of G_t. An edge $e\in E_t$ is defined as a tuple $\left(u, v, \{b_u^h, c_u^h, m_u^h\}_{h\in[1,H]}, \rho_u\right)$ where: (i) $u, v \in V_t$, (ii) the number of resource blocks (b_u^h), compute resources (c_u^h), and memory (m_u^h) is the assignment for each application h at node u such that $\sigma(h,u)=a$, with $a\in\mathcal{A}^{\mathrm{s},h} \cup \mathcal{A}^{\mathrm{b},h}$, if one or more stems or branches of the DNN are deployed on node u for application h, (iii) and ρ_u is the uplink per-resource block data rate at u. The same holds if $\sigma(h,u)=$`data`, but for the compute resources allocation which in this case is $c_u^h=0$;
- By setting K to the number of constraints plus the objective function in our optimization problem (3b)–(3g) (i.e., $K=7$), $\mathcal{F}_t=\{f_1, \ldots f_K\}$ is the set of normalized constraint attribute functions that assign to every edge $e\in E_t$ a non-negative value $f_j(e)$, with $f_j(e)$: $\left(u, v, \{ b_u^h, c_u^h, m_u^h\}_h, \rho_u\right)$ $\in E_t \to \mathbb{R}^+$, $j\in\{1,\ldots K\}$. Each $f_j(e)$ corresponds to the value of the objective function (if $j=1$) or of a normalized constraint (if $j=2,\ldots K$), setting the right hand side in the constraints expression to 1, associated with edge e.

Computational complexity. Generating the dynamic graph has low, namely, polynomial computational complexity. Indeed, we have at most $N+2=N^s+N^m+N^e+2$ vertices in the graph, hence, $O(N^2)$ edges. Each edge has $2H+1$ attributes, giving a total complexity of $O(N^2+H)$; assuming that there are more nodes than applications, the complexity can be further simplified to $O(N^2)$, i.e., quadratic in the number of nodes.

B. The QCPO solution framework

Using the QCPO approach [25], the decision process can be modeled as a constrained Markov decision process and, accordingly, the dynamic system can be defined by the tuple $<\mathcal{S}^s, \mathcal{S}^a, \mathbf{r}, \mathbf{c}, \mathbf{M}, \gamma>$, where $\mathcal{S}^s$ is the state space, $\mathcal{S}^a$ is the action space, $\mathbf{r} : \mathcal{S}^s \times \mathcal{S}^a \to \mathbb{R}$ is the reward function, $\mathbf{c} : \mathcal{S}^s \times \mathcal{S}^a \to \mathbb{R}^+$ is the cost function, $\mathbf{M} : \mathcal{S}^s \times \mathcal{S}^a \times \mathcal{S}^s \to [0,1]$ is the state transition probability, and γ is a discount factor used for computing the accumulated cost of the Markov decision process over the epochs.

In the following, we fix the time instant t and drop the dependency on t whenever clear from the context. We instead denote with τ the epoch of the QCPO process, which is performed at each t to adapt the selected configuration to the system's dynamics. The QCPO process selects the system configuration whenever there is a change in the environment (i.e., system state) to facilitate the inference requests from applications in the system. The system state is then given by the set $\mathbf{s}_t=\mathbf{s}_\tau=\{B_n, C_n, M_n, \rho_n\}_n$ and an action is represented by the set $\mathbf{a}_\tau=\{\boldsymbol{\sigma}, b_n^h, c_n^h, m_n^h\}_{h,n}$ (with $\mathbf{a}_t=\mathbf{a}_{P_{\max}}$, i.e., the action to be enacted at time t is the one selected at $\tau=P_{\max}$). QCPO implements a policy π, which selects the action maximizing the reward function (specified below). The edges (E_t) and attribute function values ($\mathcal{F}_t$) of the dynamic graph $G_t^{(\tau)}$ are updated at every epoch ($\tau\in[1, P_{\max}]$) based on system state and action.

We define the reward function as a positive (negative) inverse of energy consumption based on success (failure) in meeting the system and application constraints, given by:

$$\mathbf{r}(\mathbf{s}_\tau, \mathbf{a}_\tau) = \sum_{e\in E_t} \frac{\psi(e)}{1+f_1(e)} \tag{4}$$

$$\psi(e) = \begin{cases} 1, & \text{if } f_j(e) \le 1, j=2,\ldots K, \text{ i.e., (3b)–(3g) are met} \\ -1, & \text{otherwise.} \end{cases}$$

We further note that by selecting the edges that have a lower energy consumption (i.e., smaller f_1 in the denominator of (4)), and meet the constraint requirements, yields a higher (positive valued) reward. We also define the cost function as the weighted (weight μ_j) sum of the normalized constraint attribute functions (f_j) that correspond to the problem constraints, i.e.,

$$\mathbf{c}(\mathbf{s}_\tau, \mathbf{a}_\tau) = \sum_{e\in E_t} \sum_{j=2}^{7} \mu_j f_j(e)\,. \tag{5}$$

The estimated cumulative sum cost is given by:

$$X^{\pi}(\mathbf{s})=\sum_{\tau=0}^{\infty}\gamma^{\tau}\mathbf{c}(\mathbf{S}_{\tau},\mathbf{A}_{\tau}) \tag{6}$$

where, given π the policy function, $\mathbf{A}_{\tau}\sim\pi(\cdot|\mathbf{S}_{\tau})$ and $\mathbf{S}_{\tau}\sim\mathbf{M}(\cdot|\mathbf{S}_{\tau-1},\mathbf{A}_{\tau-1})$. Importantly, the policy π implemented by QCPO selects an action so that the quantile of the distribution of the estimated cumulative sum cost does not exceed a specified threshold (d_{th}). By doing so, it satisfies the system latency and inference quality constraints within the selected quantile.

QCPO implements a policy π maximizing the reward and limiting the quantile of the cumulative cost by leveraging two neural networks, namely, the policy and value neural networks. The former, also known as actor, is used to choose actions and update the policy; the latter, also known as critic, is used to estimate the value function. More specifically, the original optimization problem, given in (3), is modified for QCPO and expressed in terms of expected cumulative reward and quantile of the estimated cumulative cost functions, as follows:

$$\max_{\pi} V^{\pi}(\mathbf{s}_0)=\mathbb{E}_{\pi}\left[\sum_{\tau=0}^{\infty}\gamma^{\tau}\mathbf{r}(\mathbf{s}_{\tau},\mathbf{a}_{\tau})\right] \tag{7a}$$

$$\text{s.t.}\quad q_{\omega}^{\pi}\leq d_{th}, \tag{7b}$$

where ω-quantile of the random variable of the estimated cumulative sum cost is $q_{\omega}^{\pi}(\mathbf{s})=\inf\{x|\Pr(X^{\pi}(\mathbf{s})\leq x)\geq\omega\}$.

The quantile and tail probability of the cumulative sum cost is estimated using the distributional RL with the Large Deviation Principle (LDP). We use Weibull distribution (a particular case of generalized Gamma distribution) to model the desired rate of decay of the tail probability. The Weibull distribution models the tail of the estimated cumulative sum cost distribution and, hence, it can be used to compute the quantile [25]. Notice that the QCPO algorithm satisfies the quantile constraint after the training of the RL policy and value networks.

Policy and value networks. The objective function of the policy network is based on the parameter ϕ given by: $L(\boldsymbol{\phi})=\mathbf{E}[J(\boldsymbol{\phi})]$. The objective function of the value network is: $L(\boldsymbol{\Omega})=\mathbf{E}[J(\boldsymbol{\Omega})]$, where, $J(\boldsymbol{\Omega})$ represents the temporal difference error of the value function.

QCPO leverages the proximal policy optimization (PPO) algorithm [26]. It enables frequent policy optimization depending upon earlier policy and, in general, it shares the parameters between policy and value networks. By performing clipping operation on the policy probability ratio (PPR), i.e., $\mathbf{p}(\boldsymbol{\phi})=\frac{\pi_{\phi}(\mathbf{a}_{\tau}|\mathbf{s}_{\tau})}{\pi_{\phi_o}(\mathbf{a}_{\tau}|\mathbf{s}_{\tau})}$, the clipped policy objective function is obtained as: $J(\boldsymbol{\phi})=\mathbb{E}\Big[\hat{\mathcal{L}}(\mathbf{p}(\boldsymbol{\phi}))\hat{A}_{\pi_{\boldsymbol{\phi}_o}}(\mathbf{s}_{\tau},\mathbf{a}_{\tau})\Big]$ where, clipping function is defined as:

$$\hat{\mathcal{L}}(\mathbf{p}(\boldsymbol{\phi}))=\begin{cases}1-\theta, & \mathbf{p}(\boldsymbol{\phi})\leq 1-\theta\\ 1+\theta, & \mathbf{p}(\boldsymbol{\phi})\geq 1+\theta\\ \mathbf{p}(\boldsymbol{\phi}), & \text{otherwise.}\end{cases}$$

Here, $\pi_{\boldsymbol{\phi}_o}(\mathbf{a}_{\tau}|\mathbf{s}_{\tau})$ and $\hat{A}_{\pi_{\boldsymbol{\phi}_o}}(\mathbf{s}_{\tau},\mathbf{a}_{\tau})$ represent the previous policy and estimate of advantage function, respectively, while θ is the clipping parameter. $\boldsymbol{\phi}_o$ is the policy parameter prior to update. The value (quantile) advantage function is an estimate of the difference between the total weighted reward (cost) and the estimated value (quantile) function for a selected action, on the completion of an epoch. A positive value means that the chosen action is preferred. QCPO takes the policy gradient using the sum of the value and the quantile advantage [25]. The update expression of the policy network parameter, $\boldsymbol{\phi}$, for each proposed policy is given as:

$$\boldsymbol{\phi}_{t+1}=\boldsymbol{\phi}_t+\frac{\eta}{P_{\max}}\sum_{\tau=1}^{P_{\max}}\bigtriangledown_{\phi}J(\boldsymbol{\phi}). \tag{8}$$

where $\bigtriangledown_{\phi}$ represents the gradient of the policy objective function.

The overall state-based (on-policy) procedure of QIC using the QCPO action selection function is given in Algorithm 1.

Algorithm 1: QCPO Algorithm

Input: $\mathbf{s}_t$

Function I: QCPO_action_selection (s_t) **:**

1) Initialize policy neural network with random seed, $\boldsymbol{\phi}_t \in (0,1]$
2) Initialize state value function with random seed, $\boldsymbol{\Omega}\in(0,1]$;
3) **for** $\tau=1,2,\ldots P_{\max}$ **do**
   - 3.1) Observe the present state $\mathbf{s}_{\tau}$
   - 3.2) Compute current reward $\mathbf{r}(\mathbf{a}_{\tau},\mathbf{s}_{\tau})$ using (4).
   - 3.3) Set $temp=0, \mathbf{a}_{\tau}=NULL$;
   - **for** $a\in\mathcal{S}^{a}$ **do**
     - 3.4) Estimate $\mathbf{s}_{\tau+1}$ based on action a
     - 3.5) Compute $\mathbf{r}(a,\mathbf{s}_{\tau+1})$
     - **if** $\big(\mathbf{r}(a,\mathbf{s}_{\tau+1})>temp\big)$ **then**
       - $temp=\mathbf{r}(a,\mathbf{s}_{\tau+1})$;
       - $\mathbf{a}_{\tau}=a$;
     - **end if**
   - **end for**
   - 3.6) Estimate the value function $V^{\pi}(\mathbf{s}_{\tau})$ for return
   - 3.7) Estimate the quantile function $q_{\omega}^{\pi}(\mathbf{s}_{\tau}),\omega\in\{\omega_1,\omega_2,\ldots\omega_q\}$
   - 3.8) Approximate tail distribution $P_{X^{\pi}(\mathbf{s}_{\tau})}$ (right tail) using Weibull distribution, and compute advantage functions [25].
   - 3.9) Take policy gradient using sum of value and quantile advantage [25].
3) **end for**
4) $\mathbf{a}_t=\mathbf{a}_{\tau}$
5) Update policy parameter $\boldsymbol{\phi}_{t+1}$ using (8).
6) Update value parameter $\boldsymbol{\Omega}$ to maximize $L(\Omega)$.

return $\mathbf{a}_t$

Output: $\mathbf{a}_t$

Computational complexity. The QCPO action selection function has polynomial computational complexity. Indeed, we have at most $P_{\max}$ epochs and $|\mathcal{S}^{a}|$ number of actions. Given that $|\mathcal{S}^{a}|$ is $O(|\mathcal{H}|\cdot|\mathcal{N}|\cdot|\mathcal{A}^{\mathrm{s},h}|\cdot|\mathcal{A}^{\mathrm{b},h}|)$, the total complexity of QCPO is $O(P_{\max}\cdot|\mathcal{H}|\cdot|\mathcal{N}|\cdot|\mathcal{A}^{\mathrm{s},h}|\cdot|\mathcal{A}^{\mathrm{b},h}|)$.

VI. Reference Scenario

In this section, we describe the sensor-related dataset we use for our performance evaluation, as well the radio link measurements we carried out to account for real-world conditions.

Dataset and dynamic DNN model. We employ the RADIATE dataset [27], which contains data from a Navtech

CTS350-X Radar, a Velodyne HDL-32e LiDAR, and left and right ZED stereo camera for autonomous vehicle perception in diverse weather conditions such as `Sunny`, `Night`, and `Motorway`. We also consider that each mobile device is equipped with two additional LiDAR sensors, Velodyne VLS128 and Velodyne VLP16 LiDAR, in accordance with the Zenseact dataset [28]. The variety of weather conditions corresponds to different contexts associated with different levels of difficulty in achieving a satisfactory value of inference quality. Thus, this dataset presents challenges for object detection which makes it a good fit to our investigation into dynamic model deployment tailored to distinct operational constraints.

As described in Sec. III-B, our dynamic DNN integrates early fusion techniques as in [2], leveraging a ResNet-18 backbone [20]. We extended it to also include ResNet-50 and ResNet-101 as branches, thereby offering varied complexity levels for enhanced adaptability. Specifically, we partitioned the ResNet architecture, designating the initial block as the stem for each sensor modality, and then applied early fusion for merging these stems. The integrated features are then processed through the subsequent ResNet branches [2], and late fusion was added as a post-processing step only for the (less complex) ResNet 18 branches.

Each stem and branch configuration across the ResNet 18/50/101 models were trained separately to optimize performance across various detection tasks. Specifically, our architecture features six types of fusion branches – Left/Right Cameras, Lidar+Radar, and Left/Right Cameras+Lidar – each available in three complexity levels corresponding to the depths of the ResNet models, resulting in 18 branches. This design is tailored to accommodate diverse operational needs and application-specific demands.

We employ the mAP score to quantify our model's effectiveness, a popular metric that assesses object detection quality by considering precision and recall at multiple Intersection over Union (IoU) thresholds [21], [29], [27]. Initial mAP targets were set at 50% or more, a standard level in industry applications. The system's architectural complexity and the size of input data from different modalities are summarized in Tables II and III. Table II outlines the size of the raw sensory inputs and the corresponding outputs from the initial processing stage (stem). This stage's outputs serve as inputs to the various branches of the architecture. Table II also presents the computational load in terms of floating point operations (FLOP). Table III details the complexity of different architectural branches, including the total number of parameters (in millions) and the FLOPS for each branch. The architectures range from sensor-specific branches, e.g., CameraBranch18, RadarBranch18, and LidarBranch18, to more complex fusion architectures, e.g., DualCameraFusion101 and RadarLidarFusion101. The numbers '18', '50', and '101' indicate the depth of the architecture. The parameter count reflects the model's size, while the FLOPS gives insight into the computational load during inference.

Wireless link measurements. Assessing the wireless link performance is fundamental for resource allocation and for establishing a balance between computational demands and wireless communication capabilities in edge computing systems. We thus incorporate real-world wireless link traces into the evaluation of our QIC framework. To this end, we perform real-world experiments to collect performance metrics such as TCP throughput over the wireless link and MCS. We define two contexts: Outdoor and Indoor, where the Outdoor scenario is an outdoor open space with low interference from other devices and objects, and the Indoor is an environment with walls as obstacles.

We use a mobile node that travels a fixed trajectory while transmitting data through simultaneous connections through WiFi (802.11ac) for 150 s. Our mobile node is composed of a 4-wheel outdoor rover equipped with an Nvidia Jetson Nano operable through Linux OS and a WiFi antenna dongle to transmit data. In the Outdoor context (Fig. 5), the mobile node follows a circle trajectory with a radius of ~ 6 m. This circular trajectory facilitates the variation in the signal strength due to the changes in the distance between the receiver and the transmitter. In the Indoor context, instead, the mobile

TABLE II
SIZE OF STEM RAW INPUT/OUTPUT AND COMPLEXITY

Data	Stem Input	Stem Out./Branch In.	FLOPS (G)
L/R cam.	672x376	64x168x94	3.552
Radar polar	1152x1152	64x288x288	31.00
Lidar proj.	672x376	64x168x94	5.900

TABLE III
NUMBER OF PARAMETERS AND COMPLEXITY OF THE BRANCHES

Architecture	No. of parameters (M)	FLOPS (G)
CameraBranch18	40.20	21.76
RadarBranch18	40.20	115.86
LidarBranch18	40.20	23.00
DualCameraFusion18	40.28	270.8
RadarLidarFusion18	40.28	586.6
CameraLidarFusion18	40.31	286.6
LidarLidarFusion18	40.14	302.4
CameraBranch50	165.06	85.14
RadarBranch50	165.06	352.5
LidarBranch50	165.06	89.10
DualCameraFusion50	165.06	982.6
RadarLidarFusion50	165.06	2202
CameraLidarFusion50	165.06	1084
LidarLidarFusion50	165.06	1185.4
CameraBranch101	184.05	184.1
RadarBranch101	184.05	573.4
LidarBranch101	184.05	132.4
DualCameraFusion101	184.05	1496
RadarLidarFusion101	184.05	3434
CameraLidarFusion101	184.05	1562
LidarLidarFusion101	184.05	1628

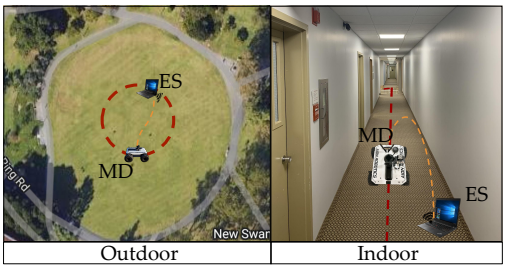


Fig. 5. Setting of the contexts used for data collection: Outdoor and Indoor.

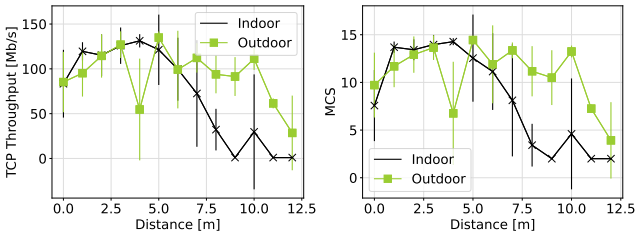


Fig. 6. Transport layer throughput (left) and MCS (right) over the radio link, for increasing distance between transmitter and receiver.

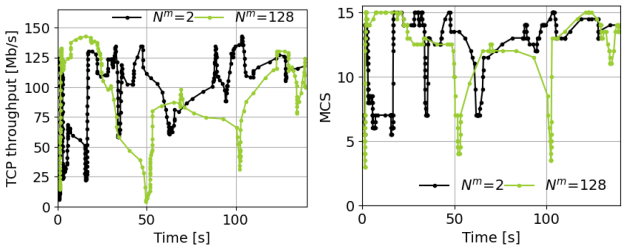


Fig. 7. Trace-plot of Transport layer throughput (left) and MCS (right) over the radio link for a small and large number of mobile nodes ($N^m = 2$ and $N^m = 128$).

node follows an L-trajectory where the node makes a turn so that multiple walls are between the transmitter and receiver. Each experiment, lasting 150 s, is characterized by a number of mobile node's TCP connections, ranging from 2 to 128 following a pattern of power of two (e.g., 2, 4, 8). Through these wireless link measurements, we approximate a scenario with multiple mobile nodes that are placed in the same area. TCP packets are generated using the iPerf3 tool, and we capture the link performance every 100 ms with *iw* [30] link status tool available in Linux OS.

Fig. 6 compares the Outdoor and Indoor contexts using two metrics: transport-layer throughput (aggregated over all the mobile node's active TCP connections) and MCS. As expected, the throughput decreases for both contexts as the distance between the ES and the mobile node grows. However, the Indoor throughput decreases faster than the Outdoor, due to the presence of obstacles between transmitter and receiver. In both cases, the throughput measurements are highly correlated with the MCS index. Further, Fig. 6 shows how the variance in the MCS value for the Indoor context is higher than in the Outdoor context. The MCS index is linked to the maximum data rate possible for a given channel bandwidth: the larger the bandwidth, the higher the data rate, hence the throughput.

The trace-plot of transport-layer throughput and MCS for a small (N_m=2) and a large (N_m=128) number of mobile nodes in the system are shown in Fig. 7. The average throughput for the mobile nodes and the average MCS is higher for a scenario with a small number of mobile nodes (N_m=2). However the average throughput drops to very low value (<1 Mbps) when the channel condition is poor and average MCS is low for the scenario with a large number of mobile nodes (N_m=128).

Importantly, the performed measurements give a good indication of the data transfer performance achieved through an OFDM-based radio interface. Specifically, the variation in the wireless channel state is well captured by the MCS index trace for the Indoor and Outdoor scenarios. For our performance evaluation of QIC, we thus compute ρ_n for the 5G NR frequency range 1 [31], [32], [33] using the MCS index measured over time and the radio parameters in Table IV. We also compute the available number of resource blocks, B_n, in the 5G NR settings as the ratio of the measured throughput to the obtained value of ρ_n.

TABLE IV
PARAMETER SETTINGS

Parameter	Value	Parameter	Value
#epochs	50	Learning rate (QCRL)	10^{-3}
Update parameter, ϱ	0.001	#neurons	300
Update rule	Adam	Steps per epoch	10,000
Tests per epoch	10	Steps per test	2,000
Channel bandwidth	50 MHz	Carrier spacing	15 KHz
Frequency	3.4 GHz	Number of data carriers	1,200

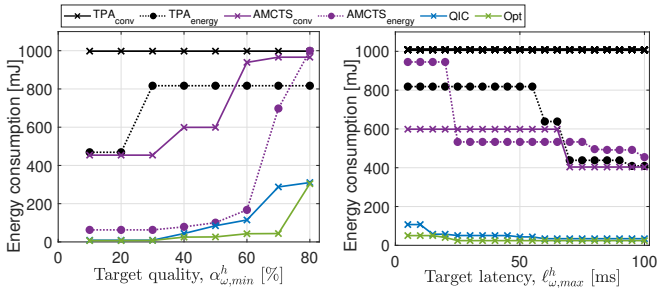
Network node resource capacity. We use mobile and edge nodes [34], respectively, associated with the following values of computational capability in trillions operations per second (TOPS) and power consumption in Watt (W): [11 TOPS, 6 W]; [153.4 TOPS, 140 W]. The power consumption of the communication interface of the mobile and edge nodes [35], [36], [37] in the idle state to maximum are in the range [3.1,3.7] W and [4096,4550] W (resp.). We consider that the memory available at the device and edge nodes for the applications is 200 kb and 4 Mb, (resp.). This reflects the operational and resource constraints at internet of things (IoT) devices while ensuring scalable deployment of such applications along with diverse and concurrent task executions at the edge [38], [39], [40]. We recall that such parameters define the computing and memory resource capacity (C_n and M_n, resp.) of the system nodes ($n \in \mathcal{N}$), and the energy consumption associated with the usage of each unit of (resp.) computational and communication resources at node n: ϵ_n^c and ϵ_n^b.

VII. Performance Evaluation

We now present the performance of QIC in a small- and a large-scale dynamic scenario, and compare it against the following benchmarks:

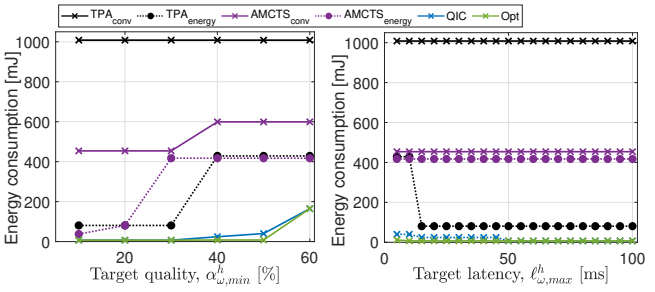
• *Two-Pass Approximation (TPA)* algorithm [17], applied on the attributed dynamic graph. It uses a greedy optimization strategy to find an efficient (cost-effective) path between the source and destination nodes under multiple constraints. The graph is traversed more than once and there are no performance guarantees; this makes it ineffective for large dynamic graph. We compare the performance of QIC with two variants, the conventional (TPA$_{conv}$ [17]) and its energy-efficient variant (TPA$_{energy}$, with cost defined in terms of energy consumption).

• *Adaptive Monte Carlo Tree Search (AMCTS)* algorithm [18], applied on the attributed dynamic graph. It is used for multi-constrained shortest temporal path selection between source and destination nodes in a graph such that it satisfies the multiple end-to-end constraints on the attributed edge weights. It combines reinforcement learning with the Monte Carlo Tree Search approach. We compare the performance of QIC against two variants, the conventional AMCTS$_{conv}$ [18]) and its energy-efficient variant AMCTS$_{energy}$, with reward and discount functions defined based on energy consumption.



(a) Target latency, $\ell^h_{\omega,\max}$ =50 ms (b) Target quality, $\alpha^h_{\omega,\min}$ =50%

Fig. 8. Small-scale scenario: Total energy consumption obtained through TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, QIC, and Opt, in the `Sunny` context, as the target inference latency and mAP quantiles vary.



(a) Target latency, $\ell^h_{\omega,\max}$ =50 ms (b) Target quality, $\alpha^h_{\omega,\min}$ =50%

Fig. 9. Small-scale scenario: Total energy consumption obtained through TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, QIC, and Opt, in the `Night` context, as the target inference latency and quality mAP quantiles vary.

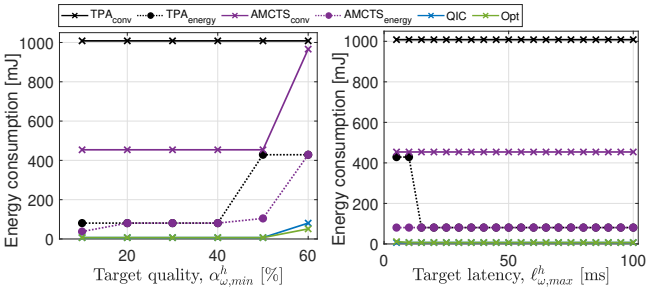
• *Optimum (Opt),* obtained through exhaustive search, considered only in the small-scale scenario where its computation is feasible.

We remark that we select [17], [18] and their energy variants because no scheme exists that specifically tackles the problem at hand.

Next, we present the results obtained considering the parameter settings listed in Table IV. We first focus on the small scenario depicted in Fig. 1(a), and then on a larger-scale scenario including multiple network nodes and applications, and three different contexts. As mentioned, in both the small-scale and large-scale scenario, the available radio resources and the MCS index that can be used by each mobile node are changing over time, as shown in Fig. 7.

Small-scale scenario. The scenario includes four data sources, three mobile nodes, two ESs, and three applications. We associate each mobile node with one application and a specific context, and we investigate how the inference quality (mAP) and latency constraints influence the energy consumption incurred by QIC and its benchmarks. We begin by looking at the performance in the case of the mobile node with the `Sunny` context. In Fig. 8(a), we fix the latency target to $\ell^h_{\omega,\max}$=50 ms with quantile ω=0.9 and vary the inference quality target $\alpha^h_{\omega,\min}$; as one might expect, a tighter inference quality target results in a larger energy consumption for all strategies. More importantly, QIC greatly outperforms TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, and AMCTS$_{energy}$, yielding savings that exceed 25%, and it almost always matches the optimum. In Fig. 8(b), we fix the mAP target to $\alpha^h_{\omega,\min}$=50% with quantile ω=0.9 and change the latency target. Besides noticing that shorter latency results in higher energy consumption, remarkably, QIC can achieve the same performance as the optimum, except when latency constraints are very tight, and consistently outperforms TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, and AMCTS$_{energy}$, on average, by 20%.

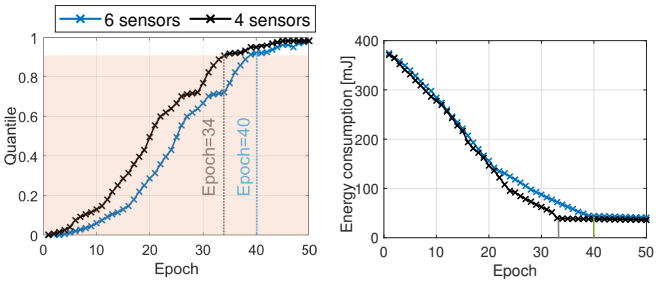
The same behavior can be observed for the `Night` and `Motorway` contexts (Fig. 9(a)–9(b) and Fig. 10(a)–10(b), resp.). Since the maximum mAP for `Night` and `Motorway` is now limited to 60% (i.e., the considered mAP requirement is less stringent), the difference between the different schemes is occasionally slightly smaller than in the `Sunny` context. Nevertheless, QIC consistently makes optimal or near-optimal decisions, while TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, and AMCTS$_{energy}$ always incurs 20% higher energy consumption relatively to QIC.



(a) Target latency, $\ell^h_{\omega,\max}$ =50 ms (b) Target quality, $\alpha^h_{\omega,\min}$ =50%

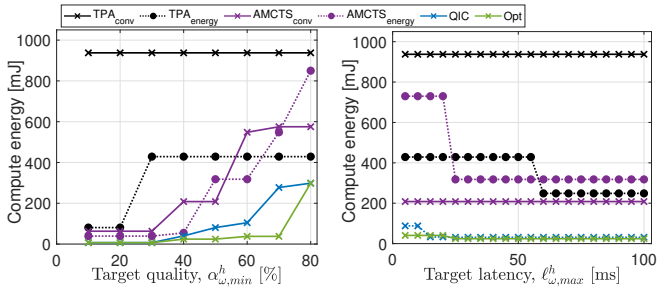
Fig. 10. Small-scale scenario: Total energy consumption obtained through TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, QIC, and Opt, in the `Motorway` context, as the target inference latency and mAP quantiles vary.

We show the learning quality evolution of the QIC algorithm during the training phase in Fig. 11. Plots report the epoch on the x-axis and the solution quality on the y-axis, expressed as constraint quantile in Fig. 11(a) and energy consumption in Fig. 11(b). We observe that QIC algorithm achieves convergence – minimizing the energy consumption while meeting the constraint quantile requirement (ω=0.9) – within 50 epochs and 70 ms of execution time on Lenovo ThinkPad P1 Gen 3 with i7-10750H CPU (2.6 GHz, 32 GB RAM). The convergence is faster, at 34 epochs instead of 40, if the state space is smaller, i.e., 4 sensors instead of 6.



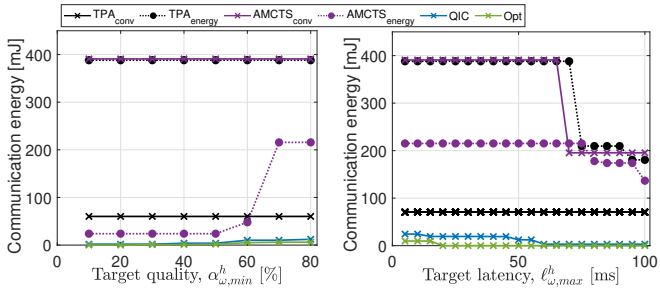
(a) (b)

Fig. 11. QIC training performance (a) Quantile (the forbidden operating region is shaded), and (b) Energy consumption, for 4 and 6 sensors at each mobile node (target latency $\ell^h_{\omega,\max}$=50 ms, target quality $\alpha^h_{\omega,\min}$=50%, ω=0.9).



(a) Target latency, $\ell^h_{\omega,\max}$ =50 ms (b) Target quality, $\alpha^h_{\omega,\min}$ =50%

Fig. 12. Small-scale scenario: Compute energy consumption obtained through TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, QIC, and Opt, in the Sunny context, as the target inference latency and mAP quantiles vary.

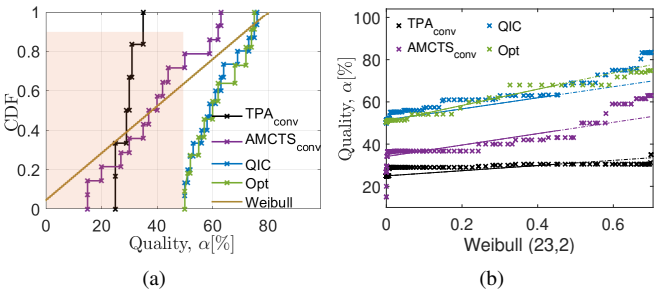


(a) Target latency, $\ell^h_{\omega,\max}$ =50 ms (b) Target quality, $\alpha^h_{\omega,\min}$ =50%

Fig. 13. Small-scale scenario: Communication energy consumption obtained through TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, QIC, and Opt, in the Sunny context, as the target inference latency and mAP quantiles vary.

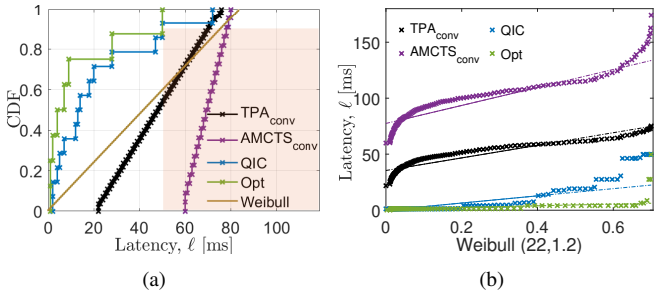
We show the compute and communication energy consumption performance for the varying latency and inference quality constraint in the Sunny context in Figures 12 and 13, respectively. We observe that the compute energy consumption is higher for higher inference quality and lower latency requirement as the best stem-branch configuration incurs higher compute requirements. Similarly the communication energy consumption increases for a lower target inference latency or a higher inference quality requirement, due to the split deployment on the mobile and edge nodes, as more complex models require transferring more data. However, the performance of QIC closely matches with the Opt and has a significantly lower compute and communication energy consumption as compared to the benchmarks, by over 50%. Indeed, even though the benchmarks TPA$_{energy}$ and AMCTS$_{energy}$ have lower compute energy consumption as compared to their conventional benchmark versions TPA$_{conv}$ and AMCTS$_{conv}$, it is significantly higher than that incurred with QIC. We further note that QIC incurs a much lower communication energy consumption than the benchmarks, by over 70%: this is due to the compute-efficient stem-branch configuration selection and energy-efficient deployment on the mobile and edge nodes that is achieved with the proposed QIC scheme.

We study the distribution of inference quality (mAP) and latency performance over the three contexts for the small-scale scenario with the inference quality requirement set as $\alpha^h_{\omega,\min}$=50%, ω=0.9, and the target latency set to $\ell^h_{\omega,\max}$=50 ms with $\omega = 0.9$. The distribution of the inference quality (mAP) and latency with TPA$_{conv}$, AMCTS$_{conv}$, QIC, and Opt schemes are shown in Figures 14(a) and 15(a), respectively. The performance of the proposed QIC and the Opt schemes always meets the requirements ($\alpha \geq 50\%$ and $\ell \leq 50\,ms$), hence, the cumulative distribution function (CDF) in the plot is always outside the shaded area (representing the infeasible operating region). TPA$_{conv}$ partially meets the latency but mostly misses the quality requirement. Conversely, AMCTS$_{conv}$ partially meets the quality but mostly misses the latency requirement. For clarity, in Figures 14 and 15 we have skipped TPA$_{energy}$ and AMCTS$_{energy}$ as these have a latency and accuracy performance poorer than TPA$_{conv}$ and AMCTS$_{conv}$, respectively.



(a) (b)

Fig. 14. Inference quality (mAP) performance (a) cumulative distribution (the forbidden operating region is shaded), and (b) Q-Q plot.

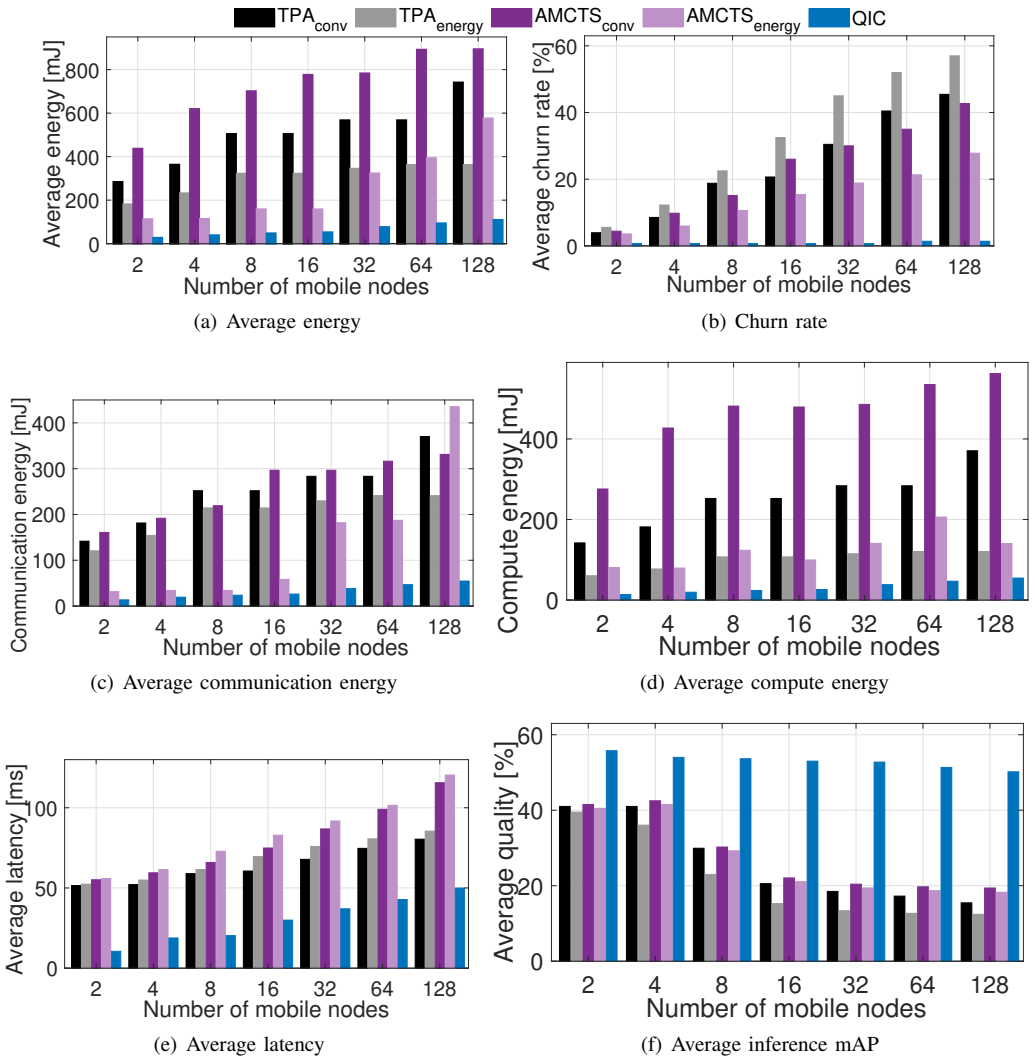


(a) (b)

Fig. 15. Latency (a) cumulative distribution (the forbidden operating region is shaded), and (b) Q-Q plot.

We use the quantile-quantile (Q-Q) plot, for the distributional assessment of the inference quality and latency performance of TPA$_{conv}$, AMCTS$_{conv}$, QIC, and Opt with respect to the Weibull approximation of the corresponding tail distribution in Figures 14(b) and 15(b), respectively. The Q-Q plot shows that the quality and latency distribution of QIC closely matches the Opt, much more than the state-of-the-art (TPA$_{conv}$, AMCTS$_{conv}$), and adhere to the desired performance limits ($\alpha \geq 50\%$ and $\ell \leq 50\,ms$). On the other hand, the Q-Q plot for the quality and latency performance of TPA$_{conv}$ and AMCTS$_{conv}$ (as well as TPA$_{energy}$ and AMCTS$_{energy}$, not shown for clarity) show a skewed distribution, with $\alpha \leq 50\%$ and $\ell \geq 50ms$, mostly missing the desired requirements.

Large-scale scenario. Next, we apply QIC to cases of large-scale scenario including multiple mobile nodes (each associated with a context and an application), multiple sensors, and ESs. Context (Sunny, Night, and Motorway) is randomly assigned to the mobile nodes with uniform probability. Here, we investigate the impact of an increasing number of mobile



(a) Average energy

(b) Churn rate

(c) Average communication energy

(d) Average compute energy

(e) Average latency

(f) Average inference mAP

Fig. 16. Large-scale scenario consisting of 10 ESs, multiple mobile nodes, and each mobile node is equipped with 6 sensors: Performance as the number of mobile nodes increases (target latency $\ell^h_{\omega,\max}$=50 ms, target quality $\alpha^h_{\omega,\min}$=50 %, $\omega = 0.9$).

nodes on the system performance. We consider $\ell^h_{\omega,\max}$=50 ms, $\alpha^h_{\omega,\min}$=50%, and ω=0.9 for all applications.

Fig. 16 compares QIC to the the benchmark schemes (TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, and AMCTS$_{energy}$) for the case of large scale scenario consisting of 10 ESs, multiple mobile nodes, and where each mobile node is equipped with 6 sensors. We consider the average performance of energy (compute and communication) and churn rate (ratio of the number of mobile nodes that fail to meet the application requirements to the total number of mobile nodes in the system), where the average value is computed over the entire duration of the simulation and for all mobile nodes. The plots in Figures 16(a)–16(b) underline, for all the schemes, the increase of both energy consumption and churn rate as the number of mobile nodes grow. This is due to an increased load on the edge servers as the number of mobile nodes increases. More interestingly, we observe the excellent energy (compute, communication, and total) performance of QIC and its effectiveness in always meeting the application requirements (mAP and latency quantiles) even with high number of mobile nodes, while the benchmark schemes (TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, and AMCTS$_{energy}$) fail to meet the requirements for a number of mobile nodes greater than two. Overall, QIC results in at least 70% reduction in energy consumption and on average 70% higher number of mobile nodes meeting their application requirements as compared to the benchmark schemes.

Figures 16(e)–16(f) shed light on why the benchmark schemes (TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, and AMCTS$_{energy}$) incurs such high churn rate, by depicting the average latency and mAP for TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, and QIC as number of mobile nodes varies. From these results, it is clear that for a number of mobile nodes larger than two, the benchmarks violates the latency requirement and it does so by an increasing margin as the number of mobile nodes increases.

Table V lists the overall execution times taken (on average) by TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, and QIC to obtain the deployment configuration in the `Sunny`, `Night`, and `Motorway` contexts of small-scale scenario as well as a case of large-scale scenario using a Lenovo ThinkPad P1 Gen 3 with i7-10750H CPU (2.6 GHz, 32 GB

TABLE V
EXECUTION-TIME [MS] TAKEN BY TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, AND QIC FOR FINDING THE DEPLOYMENT CONFIGURATION IN SMALL-SCALE (EACH CONTEXT) AND A CASE OF LARGE-SCALE SCENARIO (CONSISTING OF 10 ESS, 128 MOBILE NODES, AND EACH MOBILE NODE IS EQUIPPED WITH 6 SENSORS)

Context	TPA$_{conv}$	TPA$_{energy}$	AMCTS$_{conv}$	AMCTS$_{energy}$	QIC
Sunny	160.02	223.92	1866.57	4525.93	31.79
Night	121.12	141.39	927.08	2942.20	25.46
Motorway	107.59	124.29	883.62	2674.80	22.53
Large-scale	1281.07	1329.97	5720.40	7906.10	46.76

TABLE VI
AVERAGE ENERGY CONSUMPTION IN VARIANTS OF LARGE-SCALE SCENARIO CONSISTING OF 128 MOBILE NODES, I: 4 SENSORS, 10 ESS; II: 6 SENSORS, 10 ESS; AND III: 6 SENSORS, 20 ESS

Scen	Energy [mJ]	TPA$_{conv}$	TPA$_{energy}$	AMCTS$_{conv}$	AMCTS$_{energy}$	QIC
I	Compute	255.2	117.3	464.7	138.4	33.9
	Commun.	263.1	210.8	282.4	145.9	33.6
	Total	518.3	328.1	747.1	284.3	67.5
II	Compute	252.3	101.2	463.8	124.4	31.8
	Commun.	251.9	202.3	258.9	137.6	32.4
	Total	504.2	303.5	722.7	262.1	64.2
III	Compute	210.8	84.2	351.9	83.8	26.3
	Commun.	196.7	154.8	201.5	97.5	24.6
	Total	407.5	239.1	553.4	181.3	50.9

RAM). The execution time of QIC is at least five times lower than the benchmarks (TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, and AMCTS$_{energy}$). We emphasize that the above configuration of the experimental setup is realistic for the orchestrator implementation and the QIC solution takes less than 32 ms to run for each context in the small-scale scenario. Even in the case of large-scale scenario (consisting of 10 ESs, 128 mobile nodes, and each mobile node is equipped with 6 sensors), the QIC solution is significantly faster and takes less than 47 ms to execute.

Table VI lists the average compute, communication, and total energy consumption of TPA$_{conv}$, TPA$_{energy}$, AMCTS$_{conv}$, AMCTS$_{energy}$, and QIC in the variants of large-scale scenarios consisting of 4 or 6 sensors per mobile node along with 10 or 20 ESs and 128 mobile nodes in the system. Increase in the number of sensors or ESs reduces the energy consumption and further improves the performance of QIC with respect to the benchmarks, due to the provision of additional branches and fusion combinations to select from. Specifically, an increase in the number of sensors per mobile node as well as an increase in the number of ESs reduces the energy consumption using QIC by over 83% with respect to the benchmarks. Overall, QIC offers at least 79%, 84%, and 82% reduction in compute, communication, and total energy consumption as compared to its benchmarks.

VIII. CONCLUSIONS

We targeted dynamic scenarios where multiple AI-based applications can leverage multiple sensor data sources for their inference task. In many such scenarios, the architecture of the DNN is composed of multiple stems and branches, which can be dynamically selected so as to adapt to the available data and current conditions, and split across multiple network nodes. In this context, we proposed QIC, an effective and efficient decision-making scheme, that jointly chooses (i) the data sources and (ii) DNN sections to use, and (iii) the network nodes where each stem and branch is deployed, along with (iv) the resources to use therein. We proved that QIC has polynomial worst-case time complexity and showed, using a dynamic DNN architecture and a real-world dataset and radio link measurements, that it outperforms its benchmarks by over 70%, and closely matches the optimum.

REFERENCES

[1] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. on Mob. Comp.*, vol. 20, no. 2, 2021.

[2] A. V. Malawade, T. Mortlock, and M. A. Al Faruque, "Hydrafusion: Context-aware selective sensor fusion for robust and efficient autonomous vehicle perception," in *ACM/IEEE ICCPS*, 2022.

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

[4] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE TPAMI*, vol. 44, no. 11, 2021.

[5] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," 2017.

[6] C. Singhal, Y. Wu, F. Malandrino, M. Levorato, and C. F. Chiasserini, "Distributing inference tasks over interconnected systems through dynamic DNNs," *IEEE Transactions on Networking*, pp. 1–14, 2025.

[7] Y. Li, Y. Chen, N. Wang, and Z. Zhang, "Scale-aware trident networks for object detection," in *IEEE/CVF ICCV*, Los Alamitos, CA, USA, Nov. 2019.

[8] K. Ahmed, M. H. Baig, and L. Torresani, "Network of experts for large-scale image categorization," 2017.

[9] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert gate: Lifelong learning with a network of experts," in *IEEE CVPR*, Honolulu, HI, USA, 2017.

[10] O. Lutz *et al.*, "Escort: Ethereum smart contracts vulnerability detection using deep neural network and transfer learning," *ArXiv*, vol. abs/2103.12607, 2021.

[11] N. Shazeer, K. Fatahalian, W. R. Mark, and R. T. Mullapudi, "Hydranets: Specialized dynamic architectures for efficient inference," in *IEEE/CVF CCVPR*, Salt Lake City, UT, USA, 2018.

[12] A. V. Malawade, T. Mortlock, and M. A. A. Faruque, "Ecofusion: energy-aware adaptive sensor fusion for efficient autonomous vehicle perception," in *ACM/IEEE DAC*, 2022, p. 481–486.

[13] C. Singhal, Y. Wu, F. Malandrino, M. Levorato, and C. F. Chiasserini, "Resource-aware deployment of dynamic dnns over multi-tiered interconnected systems," in *IEEE INFOCOM*, 2024.

[14] F. Harary and G. Gupta, "Dynamic graph models," *Mathematical and Computer Modelling*, vol. 25, no. 7, pp. 79–87, Apr. 1997.

[15] F. Malandrino, C. Casetti, C.-F. Chiasserini, and M. Fiore, "Content downloading in vehicular networks: What really matters," in *IEEE INFOCOM*, 2011.

[16] Y. Li, C. Song, D. Jin, and S. Chen, "A dynamic graph optimization framework for multihop device-to-device communication underlaying cellular networks," *IEEE Wireless Comm.*, vol. 21, no. 5, pp. 52–61, Oct. 2014.

[17] A. Zhao, G. Liu, B. Zheng, Y. Zhao, and K. Zheng, "Temporal paths discovery with multiple constraints in attributed dynamic graphs," *World Wide Web*, vol. 23, 01 2020.

[18] P. Ding, G. Liu, Y. Wang, K. Zheng, and X. Zhou, "A-MCTS: Adaptive monte carlo tree search for temporal path discovery," *IEEE TKDE*, vol. 35, no. 3, 2023.

[19] C. Singhal, Y. Wu, F. Malandrino, S. G. Contreras, M. Levorato, and C. F. Chiasserini, "Resource-efficient sensor fusion via system-wide dynamic gated neural networks," in *IEEE SECON*, 2024.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[21] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *Intl. J. of Comp. Vis.*, vol. 88, 2010.

[22] W. Kang, S. Chung, J. Y. Kim, Y. Lee, K. Lee, J. Lee, K. G. Shin, and H. S. Chwa, "DNN-SAM: Split-and-merge DNN execution for real-time object detection," in *IEEE RTAS*, Milano, Italy, 2022.
[23] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *IEEE INFOCOM*, Toronto, Canada, 2020.
[24] H. Kellerer, U. Pferschy, D. Pisinger, H. Kellerer, U. Pferschy, and D. Pisinger, "Introduction to np-completeness of knapsack problems," *Knapsack problems*, 2004.
[25] W. Jung, M. Cho, J. Park, and Y. Sung, "Quantile constrained reinforcement learning: A reinforcement learning framework constraining outage probability," in *NeurIPS*, vol. 35, New Orleans, USA, Nov. 2022, pp. 6437–6449.
[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[27] M. Sheeny, E. D. Pellegrin, S. Mukherjee, A. Ahrabian, S. Wang, and A. Wallace, "Radiate: A radar dataset for automotive perception in bad weather," 2021.
[28] M. Alibeigi, W. Ljungbergh, A. Tonderski, G. Hess, A. Lilja, C. Lindstrom, D. Motorniuk, J. Fu, J. Widahl, and C. Petersson, "Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Intern. Conf. Computer Vision*, 2023.
[29] H. Ma, W. Zhao, B. Liu, and W. Chen, "Multi-scale target detection in autonomous driving scenarios based on yolov5-afam," *Applied Sciences*, vol. 14, no. 11, p. 4633, 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/11/4633
[30] Linux Wireless, "iw - nl80211 based cli configuration utility," https://wireless.wiki.kernel.org/en/users/documentation/iw, 2019.
[31] Y. Kim *et al.*, "New Radio (NR) and its evolution toward 5G-advanced," *IEEE Wir. Comm.*, vol. 26, no. 3, 2019.
[32] F. Rinaldi, A. Raschellá, and S. Pizzi, "5G NR system design: a concise survey of key features and capabilities," *Wir. Netw.*, vol. 27, no. 8, 2021.
[33] J. J. Gimenez *et al.*, "5G new radio for terrestrial broadcast: A forward-looking approach for NR-MBMS," *IEEE Trans. on Broadcasting*, vol. 65, no. 2, 2019.
[34] F. Malandrino, C. F. Chiasserini, and G. di Giacomo, "Efficient distributed DNNs in the mobile-edge-cloud continuum," *IEEE/ACM Transactions on Networking (early access)*, pp. 1–15, Nov. 2022.
[35] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," *IEEE Journal on Selected Areas in Communications*, 2016.
[36] Y. Li, A.-C. Orgerie, I. Rodero, B. L. Amersho, M. Parashar, and J.-M. Menaud, "End-to-end energy models for edge cloud-based IoT platforms: Application to data stream analysis in IoT," *Future Generation Computer Systems*, vol. 87, pp. 667–678, Oct. 2018.
[37] L. Sun, H. Deng, R. K. Sheshadri, W. Zheng, and D. Koutsonikolas, "Experimental evaluation of WiFi active power/energy consumption models for smartphones," *IEEE Transactions on Mobile Computing*, vol. 16, no. 1, pp. 115–129, Mar. 2017.
[38] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1253–1268, 2021.
[39] I. Fedorov, R. P. Adams, M. Mattina, and P. N. Whatmough, *SpArSe: sparse architecture search for CNNs on resource-constrained microcontrollers*, Red Hook, NY, USA, 2019.
[40] L. Ravaglia, M. Rusci, A. Capotondi, F. Conti, L. Pellegrini, V. Lomonaco, D. Maltoni, and L. Benini, "Memory-latency-accuracy trade-offs for continual learning on a RISC-V extreme-edge node," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, 2020, pp. 1–6.

Chetna Singhal [SM'21] is a researcher at INRIA, France. She worked as an Assistant Professor at the Indian Institute of Technology (IIT) Kharagpur from 2015 till 2024. She was an ERCIM Fellow at RI.SE, Sweden, in 2022 and a Visiting Researcher with Northeastern University, Boston, in 2018 and Politecnico di Torino, Italy, in 2015 and 2016. She completed the M.Tech. and Ph.D. at IIT Delhi in 2010 and 2015, respectively. Her research interests include machine learning orchestration, and communication networks.

Yashuo Wu is a Ph.D. student at the University of California, Irvine. She received a Master's and Bachelor's degree in Electrical and Computer Engineering from the University of Illinois, Urbana-Champaign in 2021 and 2019. Her research primarily focus on edge computing with emphasis on dynamic resources allocation, dynamic neural networks, and game theory.

Francesco Malandrino [SM'19] earned his Ph.D. degree from Politecnico di Torino in 2012 and is now a senior researcher at the National Research Council of Italy (CNR-IEIIT). His research interests include the architecture and management of wireless, cellular, and vehicular networks.

Sharon L. G. Contreras is a Ph.D. candidate in Networked Systems at the University of California, Irvine. She received a M.Sc. in Electrical Engineering from the University of Southern California (2020) and a dual B.S. in Telematics Engineering and Computer Engineering from the Instituto Tecnológico Autónomo de México (2017, 2016). Her research focuses on mobile edge computing, wireless systems, and reinforcement learning.

Marco Levorato [SM'22] is a Professor in the Computer Science department at the University of California, Irvine. He completed the PhD in Electrical Engineering at the University of Padova, Italy, in 2009. Between 2010 and 2012, he was a postdoctoral researcher jointly at Stanford and the University of Southern California. His research interests are focused on distributed computing over unreliable wireless systems, especially for autonomous vehicles and robotic applications.

Carla Fabiana Chiasserini [F'18] is a Professor at Politecnico di Torino, Italy. She worked as a visiting scholar and researcher at UCSD from 1998 till 2003. She was also a Visiting Professor at the Monash University (Australia) in 2012 and 2016, and at the Technical University in Berlin (Germany) in 2021 and 2022. She is also a Guest Professor at Chalmers University of Technology, Sweden.