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

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

C. Singhal[1], Y. Wu[2], F. Malandrino[3,4], S. Ladron de Guevara Contreras[2], M. Levorato[2], C. F. Chiasserini[5,4,6]
1: Indian Institute of Technology Kharagpur, India – 2: UC Irvine, USA – 3: CNR-IEIIT, Italy
4: CNIT, Italy – 5: Politecnico di Torino, Italy – 6: Chalmers University of Technology, Sweden

***Abstract*—Mobile systems will have to support multiple AI-based applications, each leveraging heterogeneous data sources through DNN architectures collaboratively executed within the network. 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 this end, 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 matches the optimum and outperforms its alternatives by over 80%.**

***Index Terms*—Network support to machine learning, Dynamic DNNs, Energy efficiency, 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*. 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 [1]. 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 [2]. 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 [3] 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. In contrast, **this paper represents the first contribution considering dynamic gated models in the context of layered computing/communication 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 and communication 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 accuracy 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; 4) the computation and communication 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., accuracy) 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 accuracy), but rather in terms of a target *quantile* thereof (e.g., the 90th percentile of accuracy).

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 accuracy 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 various types of 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 real-world reference scenario, including state-of-the-art DNNs and real-world wireless measurements (Sec. VI);
- In this scenario, we study QIC's performance (Sec. VII), finding it to significantly outperform state-of-the-art approaches (80% and 50% reduction in, resp., energy and application requirements failure) and closely match the optimum.

II. Related Work

Dynamic DNNs have gained popularity for adding flexibility to inference processes and enhancing both the accuracy and efficiency of deep learning-based object detection and image classification [4], [5]. 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 mobile device-cloud computation has been modeled in [6] using a directed acyclic graph. However, [6] does not consider the dynamic variation of system parameters, multiple input sources, the stem-branch architecture, or the reliability of the inference process. While [3] delves into sensor fusion, it predominantly focuses on the training phase, neglecting the inference stage. On the other hand, [7] investigates energy-efficient inference processes but does so by selecting all available inputs, resulting in suboptimal energy utilization. Energy is instead considered in [8], 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 [9] has been used,e.g., for wireless vehicular networks [10]. However, dynamic graph modeling for real-time applications requires multi-constrained temporal path discovery. In [11], this has been solved using adaptive Monte Carlo Tree Search algorithm, MCTP. In our work, we compare the performance of our proposed framework to this state-of-the-art algorithm.

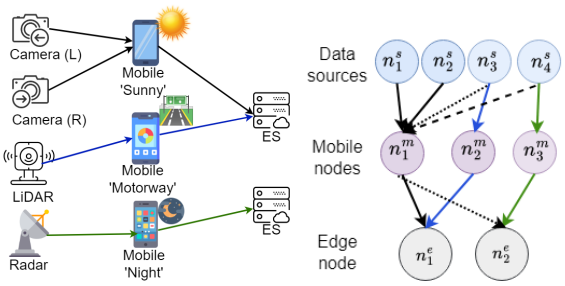
III. System Scenario

This section first introduces the system scenario we consider by characterizing the data sources and the mobile-edge continuum nodes, 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 accuracy). 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



(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.

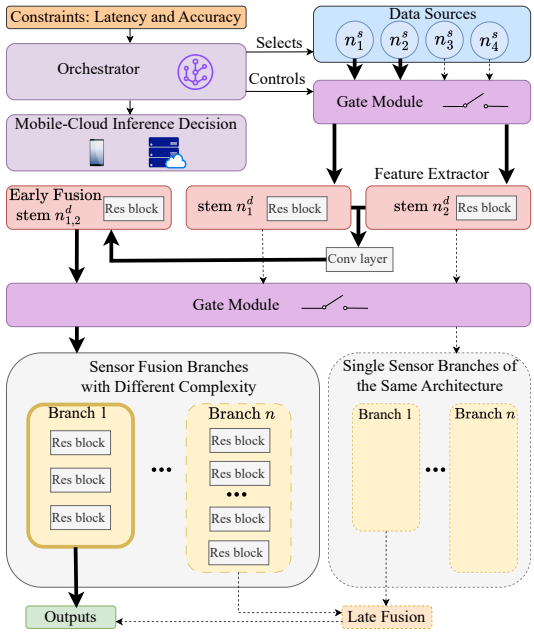


Fig. 2. Architecture implemented for object detection inference, illustrating the process where the orchestrator selects the data source, followed by the gate modules determining the appropriate stem and branch for deployment.

network 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 DNN's sections (stems or branches) to deploy locally, and generates the configuration to be used by the gates controlling the data processing throughout the DNN.

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 [3], 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 [3], which selects branches during the training phase to enhance accuracy, **our study implements the gating mechanism during the inference phase,** in order to optimize the tradeoff between output accuracy, energy expense, and latency given the current context.

Our neural model architecture processes input data from various modalities to facilitate object detection. We integrate ResNet-18/50/101 [12], our base architecture, within a faster Recursive-Convolutional Neural Network (R-CNN). 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 accuracy. 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 6 branches, each presenting 3 levels of complexity aligned with the ResNet 18/50/101 models, resulting in a total of 18 selectable branches. A second gate module then 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 accuracy. 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. Further, this interrelation heavily depends on the context. Intuitively, these quantities are quintessential to define resource allocation. We also underline how the model is splittable, i.e., 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 network nodes, $\mathcal{N}=\{\mathcal{N}^{\mathrm{s}}\cup\mathcal{N}^{\mathrm{m}}\cup\mathcal{N}^{\mathrm{e}}\}$, including: 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 $\mathcal{N}^{\mathrm{e}}$. $N=|\mathcal{N}|$, N^{s}, N^{m}, and N^{e} denote (resp.) the number of all nodes, 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 denotes the amount of computational (e.g., CPU or GPU) resources (in number of executable instructions/s) 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 accuracy 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 accuracy guarantees are required call for more robust models – even if they have more substantial resource requirements. We remark how the resource usage versus accuracy 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).

Problem formulation. 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 computation 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 $h\in$ uses node $n\in$. 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.

Concerning resource allocation, we indicate with c^h_n (in no. instructions/s) and b^h_n (in no. of resource blocks), respectively, the computational 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^h_n denotes the (outgoing) data rate available to application h at node n, i.e., $R^h_n=b_n\cdot\rho_n$.

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:

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

where ϵ^c_n and ϵ^b_n represent the energy consumption associated with the usage of each unit of (resp.) computational and communication resources at node n.

The time for sensor fusion and inference then includes two components: the computational latency and the network latency. Given application h, the former is given by:

$$\ell^h_{\mathrm{c}}(\boldsymbol{\sigma},c)=\sum_{n\in\mathcal{N}:\ \sigma(h,n)\in\mathcal{A}^{\mathrm{s},h}\cup\mathcal{A}^{\mathrm{b},h}}\frac{o^h_{\sigma(h,n)}}{c^h_n}, \tag{2}$$

while the latter is given by:

$$\ell^h_{\mathrm{b}}(\boldsymbol{\sigma},b)=\sum_{n\in\mathcal{N}:\ \sigma(h,n)=\mathsf{data}}\frac{\delta^h_n}{R^h_n}+\sum_{n\in\mathcal{N}:\ \sigma(h,n)\in\mathcal{A}^{\mathrm{s},h}\cup\mathcal{A}^{\mathrm{b},h}}\frac{\delta^h_{\sigma(h,n)}}{R^h_n}. \tag{3}$$

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: $\ell^h(\boldsymbol{\sigma},b,c)=\ell^h_{\mathrm{c}}(\boldsymbol{\sigma},c)+\ell^h_{\mathrm{b}}(\boldsymbol{\sigma},b)$, and we indicate with $\ell^h_\omega(\boldsymbol{\sigma},b,c)$ 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, $\ell^h_\omega(\boldsymbol{\sigma}, b, c)$ depends upon the distribution of the bitrate resulting from the selected MCS. Concerning accuracy, its distribution and quantiles can be estimated through several methodologies, e.g., [3], yielding the accuracy *quantile* $\alpha^h_\omega(\boldsymbol{\sigma})$.

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 orchestrator's decision problem as the following optimization problem:

$$\min_{\boldsymbol{\sigma},b,c} \epsilon(\boldsymbol{\sigma}, b, c) \tag{4a}$$

$$\text{s.t. } \alpha^h_\omega(\boldsymbol{\sigma}) \geq \alpha^h_{\omega,\min} \qquad \forall h \in \mathcal{H} \tag{4b}$$

$$\ell^h_\omega(\boldsymbol{\sigma}, b, c) \leq \ell^h_{\omega,\max} \qquad \forall h \in \mathcal{H} \tag{4c}$$

$$\sum_{h\in\mathcal{H}} c^h_n \leq C_n \qquad \forall n \in \mathcal{N} \tag{4d}$$

$$\sum_{h\in\mathcal{H}} b^h_n \leq B_n \qquad \forall n \in \mathcal{N}. \tag{4e}$$

The orchestrator seeks to minimize the total energy consumption (4a), subject to the constraints that all applications achieve their target accuracy quantile (4b) within their target application latency (4c). In doing so, the orchestrator must be mindful of the total amount of computational (4d) and radio (4e) resources available at each node. Intuitively, obtaining a better performance (thus satisfying (4b) and (4c)) tends to require more resources, but doing so would increase the energy consumption (4a). At the same time, (4d) and (4e) prevent the orchestrator from using particularly desirable (e.g., well-connected) nodes beyond their capabilities.

Property 1: The problem of minimizing (4a) subject to constraints (4b)–(4e) is NP-hard.

Proof: We prove the property via a reduction from the knapsack problem, which is NP-hard.Given an instance of the knapsack problem with a set a set $\mathcal{I}=\{i\}$ of items with weight w_i and value v_i, and given the maximum weight $w^{\max}$, we can build a *heavily simplified*, one-application version of our problem where items are mapped into data sources, the source corresponding to item i increases the energy consumption by v_i and the accuracy by w_i, the accuracy target is equal to $w^{\max}$, and selecting an item is mapped into *not* selecting the corresponding data source. Then solving our problem is equivalent to solving the knapsack problem. Finally, it can be seen by inspection that reduction is polynomial in complexity – indeed, constant, as it has no loops. ■

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

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 [13] 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.

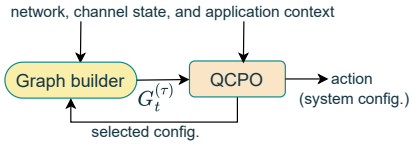


Fig. 3. QIC solution framework.

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) [13]. 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 (4a). 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 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 DNNs 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.

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

- $G_t=\{V_t, E_t, \mathcal{F}_t\}$, t=1,...T, models the system at time t;
- V_t is the set of vertices of G_t, representing data sources $n^{st}_i, i \in \{1, 2, \ldots N^s\}$, mobile nodes $n^{mt}_i, i \in \{1, 2, \ldots N^m\}$, and edge servers n^{et}_i, $i\in\{1, 2, \ldots N^e\}$, 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 $\left(u, v, \{b^h_u, c^h_u\}_{h\in[1,H]}, \rho_u\right)$ where: (i) $u, v\in V_t$, (ii) the number of resource blocks (b^h_u) and compute resources

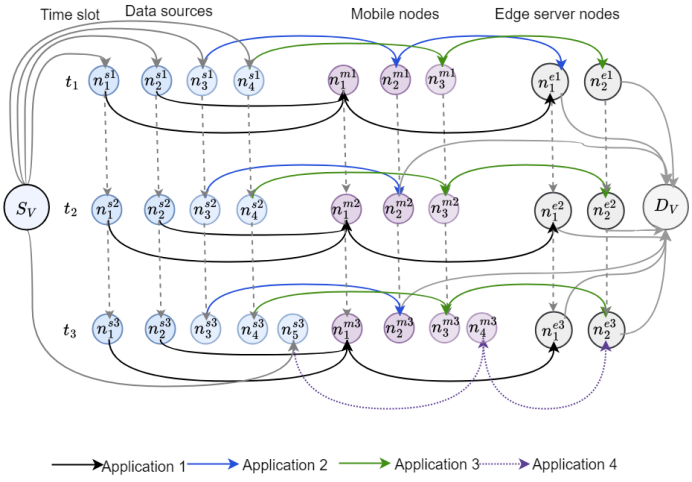


Fig. 4. Dynamic sensor fusion and inference graph over three time slots. At t_1, it reflects the example in Fig. 1; at t_3, it includes 4 applications, 5 data sources, 4 mobile nodes, and 2 edge servers.

(c_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 (4b)–(4e) (i.e., $K=5$), $\mathcal{F}_t=\{f_1,\dots f_K\}$ 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 $f_j(e)$: $\big(u,v,\{\ b_u^h,c_u^h\}_h,\rho_u\big)$ $\in E_t\to\mathbb{R}^+$, $j\in\{1,\dots K\}$. Each $f_j(e)$ corresponds to the value of the objective function (if $j=1$) or of a normalized constraint (if $j=2,\dots K$), 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, the graph has at most $N{+}2{=}N^s{+}N^m{+}N^e{+}2$ vertices, 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 [13], the decision process can be modelled 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 system state is then given by the set $\mathbf{s}_t{=}\mathbf{s}_\tau{=}\{B_n,C_n,\rho_n\}_n$ and an action is represented by the set $\mathbf{a}_\tau{=}\{\boldsymbol{\sigma},b_n^h,c_n^h\}_{h,n}$ (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:

$$\mathbf{r}(\mathbf{s}_\tau,\mathbf{a}_\tau)=\sum_{e\in E_t}\frac{\psi(e)}{1+f_1(e)} \tag{5}$$

$$\psi(e)=\begin{cases}1, & \text{if } f_j(e)\le 1, j{=}2,\dots K, \text{ i.e., (4b)–(4e) are met}\\ -1, & \text{otherwise.}\end{cases}$$

We also define the cost function as the weighted (weight μ_j) sum of the normalized constraint attribute functions that correspond to the problem constraints, i.e., $\mathbf{c}(\mathbf{s}_\tau,\mathbf{a}_\tau)=\sum_{e\in E_t}\sum_{j=2}^{5}\mu_j f_j(e)$. The estimated cumulative sum cost is: $X^\pi(\mathbf{s}){=}\sum_{\tau=0}^{\infty}\gamma^\tau\mathbf{c}(\mathbf{S}_\tau,\mathbf{A}_\tau)$ 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 $X^\pi(\mathbf{s})$ does not exceed a specified threshold (d_{th}). By doing so, it satisfies the system latency and accuracy 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 (4), is modified for QCPO and expressed in terms of expected cumulative reward and quantile of the estimated cumulative cost functions, as follows:

$$\max_\pi V^\pi(\mathbf{s}_0)=\mathbb{E}_\pi\left[\sum_{\tau=0}^{\infty}\gamma^\tau\mathbf{r}(\mathbf{s}_\tau,\mathbf{a}_\tau)\right] \tag{6a}$$

$$\text{s.t.}\quad q_\omega^\pi\le d_{th}, \tag{6b}$$

where ω-quantile of the r.v. of the estimated cumulative sum cost is $q_\omega^\pi(\mathbf{s}){=}\inf\{x|\Pr(X^\pi(\mathbf{s}){\le}x){\ge}\omega\}$. We compute the quantile and tail probability of the cumulative sum cost (modelled through the Weibull distribution) via the distributional RL with the large deviation principle [13]. QCPO meets the constraint after the RL policy and value network training.

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 uses the proximal policy optimization (PPO) [14] to enable frequent policy optimization depending on ear-

lier 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., $\mathbf{p}(\phi)=\frac{\pi_\phi(\mathbf{a}_\tau|\mathbf{s}_\tau)}{\pi_{\phi_o}(\mathbf{a}_\tau|\mathbf{s}_\tau)}$, the clipped policy objective function is: $J(\phi) = \mathbb{E}\Big[\hat{\mathcal{L}}(\mathbf{p}(\phi))\hat{A}_{\pi_{\phi_o}}(\mathbf{s}_\tau,\mathbf{a}_\tau)\Big]$ where: $\hat{\mathcal{L}}(\mathbf{p}(\phi))$ is $1-\theta$ for $\mathbf{p}(\phi) \leq 1-\theta; 1+\theta$ for $\mathbf{p}(\phi) \geq 1+\theta$; and $\mathbf{p}(\phi)$ otherwise. Here, $\pi_{\phi_o}(\mathbf{a}_\tau|\mathbf{s}_\tau)$ and $\hat{A}_{\pi_{\phi_o}}(\mathbf{s}_\tau,\mathbf{a}_\tau)$ represent the previous policy and estimate of advantage function, respectively, while θ is the clipping parameter. ϕ_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 [13]. The update expression of the policy network parameter for each proposed policy is:

$$\phi_{t+1} = \phi_t + \frac{\eta}{P_{\max}} \sum_{\tau=1}^{P_{\max}} \bigtriangledown_\phi J(\phi) \tag{7}$$

where $\bigtriangledown_\phi$ is the policy objective function gradient.

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) Init. policy neural network with rand. seed, $\phi_t \in (0,1]$
2) Init. state value function with random seed, $\mathbf{\Omega} \in (0,1]$;
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 (5)
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 $q^\pi_\omega(\mathbf{s}_\tau), \omega \in \{\omega_1, \omega_2, \ldots \omega_q\}$
3.8) Approx. right tail distrib. $P_{X^\pi(\mathbf{s}_\tau)}$ via Weibull distrib. and compute advantage functions [13].
3.9) Take policy gradient using sum of value and quantile advantage [13].
3) **end for**
4) $\mathbf{a}_t = \mathbf{a}_\tau$
5) Update ϕ_{t+1} via (7) and $\mathbf{\Omega}$ 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 $O(P_{\max} \cdot |\mathcal{H}| \cdot |\mathcal{N}| \cdot |\mathcal{A}^{\mathrm{s},h}| \cdot |\mathcal{A}^{\mathrm{b},h}|)$.

TABLE I
SIZE OF STEM RAW INPUT/OUTPUT AND COMPLEXITY

Data	Stem In.	Stem-Branch	FLOPS [G]
L/R cam.	672x376	64x168x94	3.552
Radar polar	1152x1152	64x288x288	31.00
Lidar proj.	672x376	64x168x94	5.900

VI. REFERENCE SCENARIO

In this section, we describe the sensors 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 [15], which offers data of 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`. The variety of weather conditions represents different contexts associated with different levels of difficulty in achieving a satisfactory value of inference accuracy. Thus, this dataset presents challenges for object detection across varying environmental scenarios, prompting our investigation into dynamic model deployment tailored to distinct operational constraints.

Our dynamic DNN (see Sec. III-B) integrates early fusion techniques [3], leveraging a ResNet-18 backbone [12]. We extended it to ResNet-50 and ResNet-101, thereby offering varied complexity levels for enhanced adaptability. Furthermore, 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 [3], and late fusion was added as a post-processing step only for the (less complex) ResNet 18 branches.

The system's architectural complexity and the size of input data from different modalities are summarized in Tables I and II. Table I outlines the size of the raw sensory inputs and the corresponding outputs from the initial processing stage (stem), which serve as inputs to the various branches. Table I also presents the computational load in terms of floating point operations per second (FLOPS). Table II details the complexity of different branches, including the total number of parameters (in millions) and the FLOPS for each branch. The architectures vary from specific sensory branches, e.g., CameraBranch18, RadarBranch18, and LidarBranch18, to more complex fusion architectures, e.g., DualCameraFusion101 and RadarLidarFusion101. The branches denoted with '18', '50', and '101' indicate the architecture's depth. The parameter count reflects the model's size, while the FLOPS give insight into the computational load during inference.

Radio link measurements. As radio link performance is key to establish a balance between edge computational and communication demands, we incorporate real-world radio link traces into the evaluation of QIC. Through real-world experiments, we collected TCP throughput and MCS values under two contexts: Outdoor and Indoor. The former is an outdoor open space with low interference from other devices and objects, the latter is an environment with walls as obstacles.

TABLE II
NUMBER OF PARAMETERS AND COMPLEXITY OF THE BRANCHES

Architecture	No. of param. (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
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
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

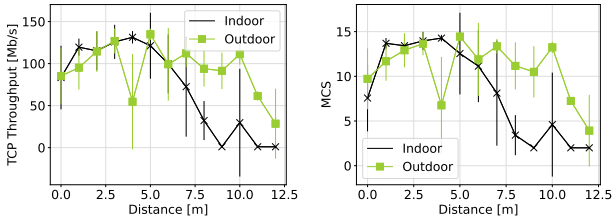


Fig. 5. Transport layer throughput (left) and MCS (right) over the radio link, for increasing distance between transmitter and receiver.

We used 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, the mobile node follows a circle trajectory with a radius of $\sim$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 case, instead, the node follows an L-trajectory so that, at the turn, 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. TCP packets are generated using the iPerf3 tool, and we capture the link performance every 100 ms with *iw* [16] link status tool available in Linux OS.

Fig. 5 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. 5 shows how the variance in the MCS value for the Indoor context is higher than in the Outdoor context. The MCS index represents the maximum

TABLE III
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	10000
Tests per epoch	10	Steps per test	2000
Channel bandwidth	50 MHz	Carrier spacing	15 KHz
Frequency	3.4 GHz	Number of data carriers	1200

data rate possible for a given channel bandwidth: the larger the bandwidth, the higher the data rate, hence the throughput.

Importantly, the performed measurements give a good indication of the data transfer performance through an OFDM-based radio interface. For QIC's evaluation, we thus compute ρ_n for the 5G NR frequency range 1 [17] using the MCS index measured over time and the radio parameters in Table III. We also compute B_n (the available no. of resource blocks) as the ratio of the measured throughput to the obtained value of ρ_n.

VII. PERFORMANCE EVALUATION

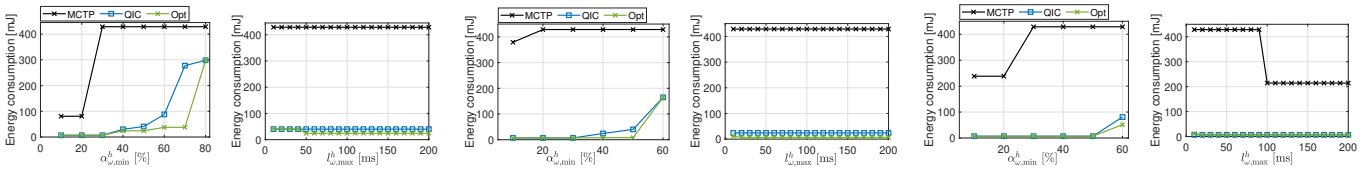
We now show QIC's performance in a small- and a large-scale dynamic scenario, against the following benchmarks:

• *Multi-Constrained Shortest Temporal Path selection (MCTP)* [11], applied on the attributed dynamic graph. It is based on an adaptive Monte Carlo Tree Search, which finds a path between a graph source and destination nodes so as to satisfy the multiple end-to-end constraints on the attributed edge weights. We select [11] because no scheme exists that specifically tackles the problem at hand.

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

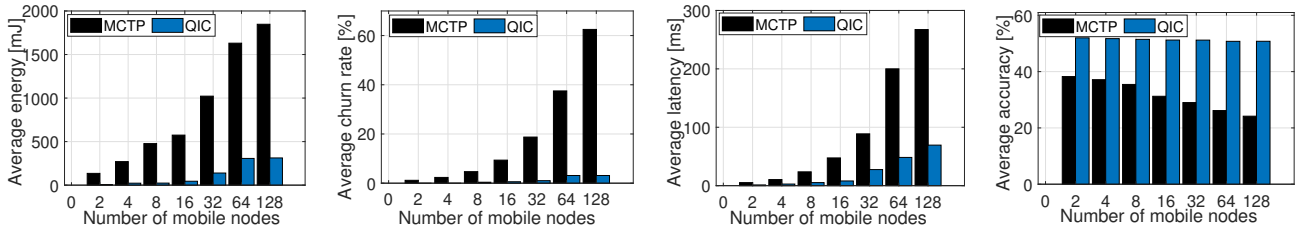
Parameters are set as listed in Table III. As mentioned, in both the small-scale and large-scale scenario, the available radio resources and the MCS index that can be used by each mobile node are changing over time.

Small-scale scenario. The scenario, depicted in Fig. 1(a), includes 4 data sources, 3 mobile nodes, 2 ESs, and 3 applications. We associate each mobile node with one application and a specific context, and investigate the impact on energy consumption of accuracy and latency constraints. We begin by looking at the performance in the case of the mobile node with the `Sunny` context. In Fig. 6(a), we fix the latency target to $\ell^h_{\omega,\max}$=50 ms with quantile ω=0.9 and vary the accuracy target $\alpha^h_{\omega,\min}$; as one might expect, a tighter accuracy target results in a larger energy consumption for all strategies. More importantly, QIC greatly outperforms MCTP, yielding savings that exceed 25%, and it almost always matches the optimum. In Fig. 6(b), we fix the accuracy target to $\alpha^h_{\omega,\min}$=50% with quantile ω=0.9 and change the latency target (again, MCTP cannot meet the target accuracy and latency with higher ω's). 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 MCTP, on average, by 80%. The same behavior can be observed for the `Night` and `Motorway` contexts (Figs. 6(c)–6(d)



(a) $\ell^h_{\omega,\max}$ =50 ms (b) $\alpha^h_{\omega,\min}$ =50% (c) $\ell^h_{\omega,\max}$ =50 ms (d) $\alpha^h_{\omega,\min}$ =50% (e) $\ell^h_{\omega,\max}$ =50 ms (f) $\alpha^h_{\omega,\min}$ =50%

Fig. 6. Small-scale scenario: Total energy consumption obtained through MCTP, QIC, and Opt, in the `Sunny` (a,b), `Night` (c,d), and `Motorway` (e,f) context, as the target inference latency and accuracy quantiles vary.



(a) Average energy (b) Churn rate (c) Average latency (d) Average accuracy

Fig. 7. Large-scale scenario: Performance as the number of mobile nodes increases ($\ell^h_{\omega,\max}$=50 ms, $\alpha^h_{\omega,\min}$=50 %, $\omega = 0.9$).

and Figs. 6(e)–6(f), resp.). Since the maximum accuracy for `Night` and `Motorway` is now limited to the less stringent requirement of 60%, 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 MCTP always incurs 25% higher energy consumption relatively to QIC.

Large-scale scenario. Next, we apply QIC to a scenario with multiple mobile nodes and 10 ESs. Context (`Sunny`, `Night`, and `Motorway`) and applications are uniformly assigned to mobile nodes. We set $\ell^h_{\omega,\max}$=50 ms, $\alpha^h_{\omega,\min}$=50%, and ω=0.9 for all applications (for larger ω's MCTP cannot meet the requirements). Fig. 7 compares QIC to the benchmark, in terms of average performance of energy and churn rate (ratio of the no. of mobile nodes that fail to meet the application requirements to the total number of mobile nodes in the system), with the average value computed over the entire duration of the radio link measurement trace and the number of mobile nodes. Figs. 7(a)–7(b) underline the increase of the above metrics as the number of mobile nodes, hence the load on the edge servers, grows. More interestingly, QIC has excellent energy performance and always meets the application target accuracy and latency quantiles, even with high number of mobile nodes, whereas MCTP fails to meet these requirements for more than 8 mobile nodes. Overall, QIC results in more than 80% reduction in energy consumption and on average 70% higher number of mobile nodes meeting their application requirements as compared to MCTP. Figs. 7(c)–7(d) shed light on MCTP's high churn rate, by depicting the average latency and accuracy for both QIC and MCTP. It is clear that for more than 8 mobile nodes, MCTP violates the latency requirement and it does so by a large margin.

VIII. Conclusions

We targeted dynamic scenarios where multiple DNN-based applications can leverage multiple data sources for their inference task. Given DNN's architecture with multiple stems and branches that can be dynamically selected, we proposed QIC to jointly choose (i) the input data sources and (ii) the DNN sections to use, and (iii) the nodes where each stem and branch is deployed, along with (iv) the resources to use therein. We proved QIC's polynomial worst-case time complexity and, using a dynamic DNN architecture, a real-world dataset and radio link measurements, showed that QIC closely matches the optimum and outperforms its benchmarks by over 80%.

Acknowledgments

This work was supported by the European Union through Grant No. 101139266 (6G-INTENSE project) and the Next Generation EU under the Italian NRRP, M4C2, Investment 1.3, CUP E13C22001870001, PE00000001 - program "RESTART".

References

[1] 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.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] F. Harary and G. Gupta, "Dynamic graph models," *Mathematical and Computer Modelling*, vol. 25, no. 7, 1997.

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

[11] 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.
[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.
[13] W. Jung, M. Cho, J. Park, and Y. Sung, "Quantile constrained reinforcement learning: A reinforcement learning framework constraining outage probability," in *NeurIPS*, 2022.
[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, arXiv:1707.06347.
[15] M. Sheeny and et al., "Radiate: A radar dataset for automotive perception in bad weather," in *IEEE ICRA*, 2021.
[16] Linux Wireless, "iw - nl80211 based cli configuration utility," https://wireless.wiki.kernel.org/en/users/documentation/iw, 2019.
[17] 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.