Energy-Efficient Dynamic Training and Inference for GNN-Based Network Modeling

Chetna Singhal, Yassine Hadjadj-Aoul

Energy-Efficient Dynamic Training and Inference for GNN-Based Network Modeling

Chetna Singhal
Inria, Univ Rennes, CNRS, IRISA
Rennes, France
chetna.iitd@gmail.com

Yassine Hadjadj-Aoul
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
yassine.hadjadj-aoul@irisa.fr

***Abstract*—Efficient network modeling is essential for resource optimization and network planning in next-generation large-scale complex networks. Traditional approaches, such as queuing theory-based modeling and packet-based simulators, can be inefficient due to the assumption made and the computational expense, respectively. To address these challenges, we propose an innovative energy-efficient dynamic orchestration of Graph Neural Networks (GNN) based model training and inference framework for context-aware network modeling and predictions. We have developed a low-complexity solution framework, QAG, that is a Quantum approximation optimization (QAO) algorithm for Adaptive orchestration of GNN-based network modeling. We leverage the tripartite graph model to represent a multi-application system with many compute nodes. Thereafter, we apply the constrained graph-cutting using QAO to find the feasible energy-efficient configurations of the GNN-based model and deploying them on the available compute nodes to meet the network modeling application requirements. The proposed QAG scheme closely matches the optimum and offers atleast a 50% energy saving while meeting the application requirements with 60% lower churn-rate.**

***Index Terms*—Network modeling, Adaptive GNN Training and Inference, Machine learning orchestrator, Energy-efficiency, Mobile-edge-cloud continuum**

I. Introduction

Accurate network modeling is essential for applications like network digital twins (NDT), resource optimization, and traffic flow control in complex next-generation networks. While packet-level simulators can provide high accuracy, they come with significant computational costs. Network simulators like ns3 and Omnet++ may require considerably long simulation times (several hours) depending on the network and the traffic flow characteristics [1], [2]. Furthermore, methods such as the Markov model, queuing theory, and network simulators have been employed for these tasks, but they introduce assumption-based limitations and increased computational costs.

Recently, machine learning has been applied for efficient data-driven network modeling. Graph Neural Networks (GNN) can learn complex non-linear behavior in networks and predict QoS parameters with a level of accuracy similar to computationally expensive packet-level simulators [2]. GNNs have the potential for accurate network delay and jitter prediction in future networks, even with complex topologies [3], [4]. However, the GNN model training can be computationally expensive and time-consuming [4]. The machine learning inference performance depends highly on the model architecture and computation platform [5], [6]. The existing works do not deal with application-awareness or dynamic network modeling using GNNs, which are essential for energy-efficient orchestration in the mobile-edge-cloud continuum network.

Our work lies at the intersection of two major fields, namely, machine learning for network modeling and dynamic machine learning. In this paper, we have developed an energy-efficient dynamic GNN orchestration framework for efficient network modeling, as illustrated in Fig. 1. Network modeling applications require delay, jitter, and packet-loss predictions for large-scale networks with a required level of inference latency and quality (loss). The *orchestrator* uses such context information to dynamically model large-scale networks. The mobile-edge-cloud continuum system consists of compute nodes, such as, Central Processing Unit (CPU), Graphical Processing Unit (GPU), and Tensor Processing Unit (TPU). These nodes handle the GNN model training/update and inference for heterogeneous applications. We have developed a dynamic network modeling orchestrator that decides whether to deploy a pre-trained model or update/train it using a smaller-network data source such that the application inference requirements are met. In doing so, it also determines the compute node over which the GNN based model executes.

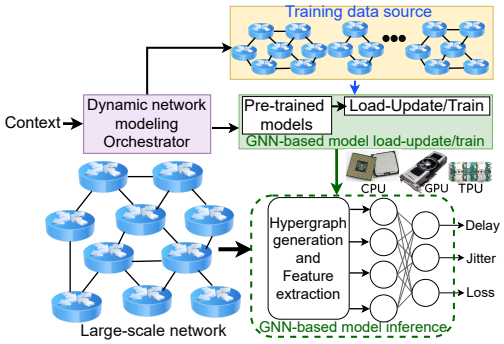


Fig. 1: Dynamic orchestration of GNN-based model training and inference for network modeling and prediction.

The main contributions of this work are given as follows:

1) We model the energy consumption and latency performance of the training-update-inference task of the GNN-based network modeling and optimization applications.
2) We minimize the energy consumption at the system level to deploy network modeling applications, while meeting their quality (loss) and latency requirements.
3) We design an energy-efficient orchestrator that dynamically selects the extent of train-update-infer function to meet the application requirements and system constraints.
4) We model the system using a tripartite graph and develop

a low-complexity solution, QAG, using the Quantum approximation optimization to solve the above problem.

II. System Model

We consider a computing framework composed of CPUs, GPUs, and TPUs that dynamically trains GNN models to predict QoS parameters. The objective of the overall system is to support a set of such network modeling applications whose central component is a set of GNN models that are trained (with updates) using the small-network information and thereafter perform predictions using the large-scale network information to meet the required application requirements.

The edge of the network infrastructure hosts a network modeling orchestrator that possesses knowledge of the applications and capabilities of the computing nodes' resources, network datasets, and configurations. Formally, the main elements defining the system are:

• A set $\boldsymbol{\sigma}$ of GNN model configurations indexed with $\sigma \in \{1, \ldots, S\}$, composed of a predefined set of stages in the message-passing architecture and a data source. The data sources contain small-scale network information that serves as input to the training-update module and facilitates the GNN model for network predictions on the large-scale network for applications. Each configuration is defined as a tuple $\sigma = \{\texttt{data\_source}, \texttt{epochs}, \texttt{steps\_per\_epoch}, \psi\}$,

$$\text{where, } \psi = \begin{cases} 0, \text{ load pre-trained model \& infer} \\ 1, \text{load pre-trained model, update, \& infer} \end{cases}. \quad (1)$$

• A set $\mathcal{H}$ of applications $h \in \{1, \ldots, H\}$. Each application has specific requirements defined as the target inference quality (e.g., target loss) ℓ^h_{max}, and the maximum inference latency τ^h_{max}. In the following, we refer to the applications and the GNN representing their essential components interchangeably.

• A set $\mathcal{N}$ of computationally-capable nodes, including (i) GPUs, (ii) CPUs, and (iii) TPUs.

Given an application (context) h, the orchestrator determines which configuration of the GNN model should be used, and to what extent should it train-update-infer. It assigns these models to the computation nodes (CPUs, GPUs, and TPUs) in such a way that the application inference requirements are fulfilled. Note that a computing node may be allocated zero, one, or multiple GNN models, which are all executed.

We represent the overall system by means of an *application-load-resource weighted complete tripartite graph* model [7] with three sets of vertices, i.e., *application*, *GNN model configurations (data source and architecture)*, and *computing nodes*. We establish a relationship between the applications' requirements (quality and latency) in $\mathcal{H}$, the compute load of configurations in $\boldsymbol{\sigma}$, and the resources made available by the heterogeneous computing nodes in $\mathcal{N}$. It matches resources handled by the system, i.e., offered by the computing nodes and the computing load required by the applications.

We show the tripartite-graph representation for a small-scale scenario in Fig. 2. It consists of two applications, four possible GNN-based model configurations, and two compute nodes.

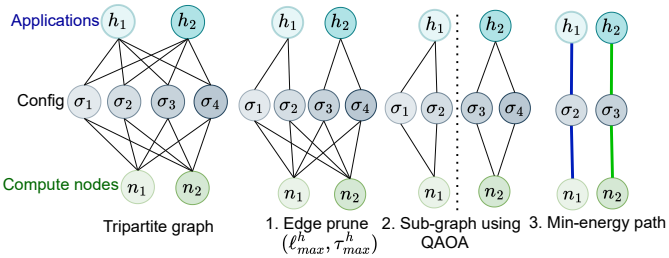


Fig. 2: Application-load-resource tripartite graph model. This small-scale example shows orchestration steps of two applications (blue and green edges) in a system with four configs. (combination of two GNN architectures and two data sources), and two computing nodes (CPU and GPU).

[s,t]	c^h	ℓ^h
h_1, σ_1	50	15
h_1, σ_2	30	25
h_1, σ_3	20	40
h_1, σ_4	10	50
h_2, σ_1	100	65
h_2, σ_2	80	75
h_2, σ_3	70	20
h_2, σ_4	60	30

[s,t]	c^1, c^2	ℓ^1, ℓ^2
σ_1, n_1	50,100	15,65
σ_2, n_1	30,80	25,75
σ_3, n_1	20,70	40,20
σ_4, n_1	10,60	50,30
σ_1, n_2	50,100	15,65
σ_2, n_2	30,80	25,75
σ_3, n_2	20,70	40,20
σ_4, n_2	10,60	50,30

Fig. 3: Edge weight for application ($h \in \mathcal{H}$, $v_i \in \mathcal{V}_1$) to configuration ($\sigma \in \boldsymbol{\sigma}$, $v_j \in \mathcal{V}_2$) (left) and configuration to compute node ($n \in \mathcal{N}$, $v_k \in \mathcal{V}_3$) vertices (right) in the tripartite graph.

More formally, we denote the graph, illustrated in Fig. 2, with $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V}$ are the vertices and $\mathcal{E}$ are the edges. The vertices $V_1 \subset \mathcal{V}$ correspond to the heterogeneous applications $h_i \in \mathcal{H}$. The vertices $V_2 \subset \mathcal{V}$ correspond to the GNN model configurations $\sigma_j \in \boldsymbol{\sigma}$. The vertices $V_3 \subset \mathcal{V}$ correspond to the system computing nodes $n_k \in \mathcal{N}$. A resource sharing setting is in place, where applications and corresponding selected configurations are assigned separate portions of computing resources on selected computing nodes.

The computational resource requirement (in operations) and inference quality (loss) associated with the edge connecting vertices v_i and v_j for application h are indicated by $c^h(v_i, v_j)$ and $\ell^h(v_i, v_j)$, resp. We associate each edge in $\mathcal{E}$ with a multidimensional weight $[c^h(v_i, v_j), \ell^h(v_i, v_j)]$. The edge weights for the small-scale example (in Fig. 2) are given in Fig. 3.

We use the following indicator variable that denotes whether the orchestrator selects the corresponding vertices of the edge to be active in the system.

$$\chi^h(v_i, v_j) = \begin{cases} 1, & \text{if } v_i \text{ and } v_j \text{ are active for application } h \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

The set containing the indicator variable values for the tripartite graph is denoted as $\boldsymbol{\chi}$. The latency of the train-update-inference task for application $h \in \mathcal{H}$ is defined as:

$$\tau^h = \sum_{\forall v_i, v_j \in \mathcal{V}, v_i \neq v_j} t^h(v_i, v_j) \cdot \chi^h(v_i, v_j),\ t^h(v_i, v_j) = \frac{c^h(v_i, v_j)}{x^h_c(v_j)} \quad (3)$$

where, $x^h_c(v_j)$ is the allocated compute resources [in operations per second] to application h at compute node $v_j \in V_3$. We denote the required resource by a configuration $v_i \in V_2$ for application h deployed on compute node $v_j \in V_3$ as $c^h(v_i, v_j)$. The value of $x^h_c(v_j) = \infty, \forall v_j \in V_1 \cup V_2$, i.e. there is no compute resource limit on the configuration vertices. Energy consumption [in J] for application h is defined as:

$$E^h = \sum_{\forall v_i, v_j \in \mathcal{V}, v_i \neq v_j} t^h(v_i, v_j) \cdot \chi^h(v_i, v_j) \cdot \xi(v_j) \quad (4)$$

where, $\xi(v_j)$ [in W] is the power consumption of the compute node $v_j \in V_3$ and $\xi(v_j) = 0$, $\forall v_j \in V_1 \cup V_2$.

III. Context-aware GNN-based Network Modeling

A. Problem formulation

The optimization problem for the orchestrator is:

$$\min_{\boldsymbol{\chi}} \sum_{h \in \mathcal{H}} E^h \quad (5a)$$

$$\text{s.t. } \tau^h \leq \tau^h_{max} \quad (5b)$$

$$\sum_{\forall v_i \in V_1, \forall v_j \in V_2} \ell^h(v_i, v_j) \cdot \chi^h(v_i, v_j) \leq \ell^h_{max}, \ \forall h \in \mathcal{H} \quad (5c)$$

$$\sum_{\forall v_i \in V_1, \forall v_j \in V_2} \chi^h(v_i, v_j) \cdot c^h(v_i, v_j) \leq \sum_{\forall v_i \in V_2, \forall v_j \in V_3} \chi^h(v_i, v_j) \cdot x^h_c(v_j), \forall h \in \mathcal{H} \quad (5d)$$

$$\sum_{\forall v_i \in V_1, \forall v_j \in V_2} \chi^h(v_i, v_j) = 1, \sum_{\forall v_i \in V_2, \forall v_j \in V_3} \chi^h(v_i, v_j) = 1, \ \forall h \in \mathcal{H} \quad (5e)$$

The objective (5a) of the orchestrator is to minimize the overall energy consumption in the system to deploy GNN models, and facilitate the application set $\mathcal{H}$, subject to latency (5b), quality-loss (5c), and compute-resource constraints (5d). This is achieved by optimizing the values of the indicator variables in the set, $\boldsymbol{\chi}$, for the tripartite graph representation of the system. Specifically the constraint (5d) ensures that the compute requirements of the configuration that is selected for an application $h \in \mathcal{H}$ is met by the resulting resource allocation at compute nodes. The constraint (5e) ensures that only one configuration is selected for each application and only one compute node completely deploys the GNN based model for that application, i.e. there is no split in the train-update-inference of the model.

Property 1: The optimization problem (5a) subject to constraints (5b)–(5e) is NP-hard.

Proof: We reduce a known NP-hard problem, the Steiner tree problem (STP) [8], to a *simplified* instance of our problem, defined in (5a)–(5e). Given an instance of the STP, we construct a corresponding instance of the problem by creating: a selected configuration for all the application vertices to connect except one; one GNN configuration, corresponding to the remaining vertex to connect; compute nodes for all configuration vertices in the STP instance; and only one of the compute nodes has enough capabilities to run the GNN configuration. Also, the deployment of the configurations on the compute nodes reproduces that of the STP instance, and one component of the weights in our problem instance is set to match the weights in the STP instance while all others are set to zero. Solving our problem to optimality thus yields an optimal solution to the STP instance. Hence, the two problems are equivalent. Since the reduction takes polynomial (linear) time (each edge and vertex of the STP instance is processed once) and the STP problem is NP-hard [8], the NP-hardness of our problem is proved. ■

It is also worth noting that the instance of our problem created in the proof above is very simple, and that on top of being NP-hard, our problem is significantly *more* complex than STP. Hence, we propose below an algorithmic solution, leveraging a quantum approximate optimization approach applied to the graph representation that, efficiently and very conveniently, finds the feasible decisions that meet the application requirements and system constraints.

B. Solution framework: QAG

We propose the solution framework QAG, Algorithm 1, that performs Quantum Approximate Optimization (QAO) for adaptive orchestration of GNN-based network modeling in the mobile-edge continuum systems. It consists of three functions, namely, `Edge_prune`, `Sub_graph_using_QAOA`, and `Min_energy_path`, performing the steps illustrated in the example that is shown in Fig. 2. The `Edge_prune` function removes the infeasible edges (that violate the loss and latency requirement) from $\mathcal{G}$.

Algorithm 1: QAG: QAO for Adaptive orchestration of GNN-based network modeling

1 **Input:** $\mathcal{G}$, τ^h_{max}, ℓ^h_{max};
2 **Function I: Edge_prune** $(\mathcal{G}, \ell^h_{max}, \tau^h_{max})$ **:**
3 **Remove infeasible edges**
4 **for** *each vertex* $v \in \mathcal{V}$ *and each edge* $(v, v') \in \mathcal{E}$ **do**
5 **if** $\ell^h(v, v') > \ell^h_{max}$ *or* $t^h(v, v') > \tau^h_{max}$ **then**
6 Remove edge (v, v'): $\mathcal{E} = \mathcal{E} \backslash (v, v')$
7 **end**
8 **end**
9 **return** *Updated graph:* $\mathcal{G}$
10 **Function II: Sub-graph_using_QAOA** $(\mathcal{G})$ **:**
11 **1. Create complement graph**
12 **for** *each vertex* $v \in \mathcal{V}$ **do**
13 Include v in $\mathcal{V}'$
14 **for** *each vertex* $v' \in \mathcal{V}$ **do**
15 **if** $v \neq v'$ *and edge* $(v, v') \notin \mathcal{E}$ **then**
16 Include edge (v, v') in $\mathcal{E}'$
17 **end**
18 **end**
19 **end**
20 **2. QAOA**$(\mathcal{G}' = \{\mathcal{V}', \mathcal{E}'\})$
21 Create QAOA circuit $\mathcal{Q}$ using parameters in Table I
22 Simulate $\mathcal{Q}$ using parameters in Table I
23 Optimize $\mathcal{Q}$ to minimize objective function value
24 **3. Create Sub-graphs**
25 $\mathcal{V}_j = \{i | q_i = j - 1, i \in [1, N]\}$, $j \in \{1, 2\}$
26 **if** $\mathcal{V}_j$ *has a single application vertex* **then**
27 Include $\mathcal{G}_j$ in $\widetilde{\mathcal{G}}$
28 **else**
29 Repeat lines 13-30 on $\mathcal{G}_j$
30 **end**
31 **return** *Graph subsets:* $\widetilde{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_H\}$
32 **Function III: Min_energy_path** $(\widetilde{\mathcal{G}})$ **:**
33 **for** *each application* $h \in \mathcal{H}$ **do**
34 $\chi^h(v_i, v_j) = 1$, for all (v_i, v_j) that $\min E^h, \forall v_i, v_j \in V_h$.
35 **end**
36 **return** $\boldsymbol{\chi}$ as QAG output

In `Sub_graph_using_QAOA` function we first define the complement of the tripartite graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ as $\mathcal{G}' = \{\mathcal{V}', \mathcal{E}'\}$, where $\mathcal{V}' = \mathcal{V}$ and $\mathcal{E}' = \mathcal{K} \backslash \mathcal{E}$, $\mathcal{K}$ consists of all two element subsets of $\mathcal{V}$, and $\mathcal{K} \backslash \mathcal{E}$ is the relative complement of $\mathcal{E}$ in $\mathcal{K}$. Thereafter we apply the QAOA max-cut [9], [10]

on the complement graph $\mathcal{G}'$ with the constraint that the subset graphs should have at least one vertex from each vertex set, $\mathcal{V}_1, \mathcal{V}_2$, and $\mathcal{V}_3$. The objective function to obtain best subsets using QAOA is defined as:

$$\min_{q\in\{0,1\}^N} \sum_{\substack{1\le i\le N, 1\le j\le N \\ i\neq j,(i,j)\in\mathcal{E}'}} -\big((q_i\cdot(1-q_j)+((1-q_i)\cdot q_j)\big) \quad (6)$$

where, N is the number of vertex in the graph. We optimize the p-layer QAOA circuit [9] based on the above objective. QAOA circuit with the Since we apply the QAOA max-cut on the complement graph, the above objective function minimization happens when the nodes connected by the edges in the complement graph are assigned to separate subsets. In terms of feasibility all the compute node vertices are equivalent due to pruning step and the resulting quantum states might reflect the subset graphs similarly. The highest probability quantum state $q = |q_1 q_2 \dots q_N\rangle, q_i \in \{0,1\}, 1 \le i \le N$, gives the two subgraphs with vertices $V_j = \{i|(q_i = j-1)\}, j \in \{1,2\}$. In case, the subgraph has more than one application vertex then further subgraphs are obtained from it using the above approach. Finally, the `Min_energy_graph` finds the minimum energy path for each application from its corresponding sub-graph and assigns the indicator variable (χ^h) accordingly.

TABLE I: Parameter settings

Quantum parameter	**Value**	**GNN-model parameter**	**Value**
Quantum gate (γ_0, β_0)	(-1,-3)	Learning rate	10^{-3}
Number of layers p, shots	2, 100	Message-passing iter.	8
Number of iterations	100	Update rule	Adam

TABLE II: Application, scenario, predicted parameter, and data sources

App.	**Scenario**	**Predict par.**	**Train/Test network data**
h_1	Real-traffic	Delay	GERMANY50, NOBEL-GBN, GEANT, ABILENE
h_2	Traffic models	Delay	GBN, GEANT, NSFNET
h_3	Traffic models	Jitter	GBN, GEANT, NSFNET
h_4	Traffic models	Packet-loss	GBN, GEANT, NSFNET
h_5	Scheduling	Delay	GBN, GEANT, NSFNET
h_6	Scheduling	Jitter	GBN, GEANT, NSFNET
h_7	Scheduling	Packet-loss	GBN, GEANT, NSFNET

TABLE III: Compute nodes Power usage [W] and compute capacity [TOPS]

Compute Node	**Compute Type**	**Power [W]**		**TOPS**
		Idle	Max	
n_1	CPU	5	12	2
n_2	T4 GPU	36	70	80
n_3	TPU v2	53	280	180

The quantum approximation optimization algorithm offers a polynomial-level complexity [11]. The initialized QAO for max-cut with p layers achieves a $O[poly(p)]$ complexity [9]. For a N node tripartite graph, the QAG algorithm with a two-layer architecture exhibits $O[N + Hpoly(2)]$ complexity which is significantly lesser than the $O[N^N]$ complexity of the brute-force optimal exhaustive-search method.

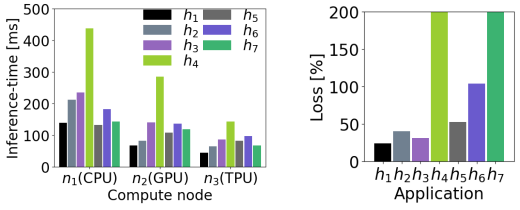
IV. Performance Evaluation

In this section, we describe the reference scenario and examine the impact of different GNN model configurations on latency, quality, and energy cost. Then, we introduce the benchmarks against which we compare QAG, and present the applications' performance in the reference system.

A. Reference scenario

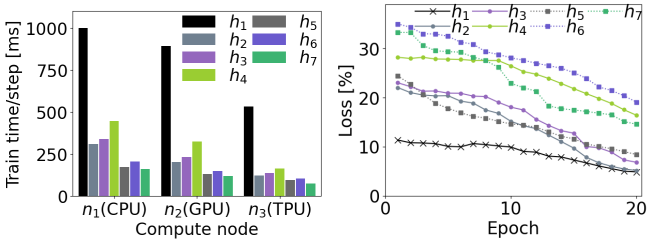
GNN-based model architecture. The network is defined by three main components: active flows in the network, queues at each output port of the network devices, and physical links. A 32 element hidden state vector encodes the initial features of these components by using a 2-layer fully-connected neural network with ReLU activation functions and 32 units each. This is followed by a three-stage message passing algorithm that combines and updates the state of the components over 8 iterations. Finally, the readout functions are implemented as 3-layer fully-connected neural networks with a ReLU activation function for the hidden layers and a linear one for the output layer. These are applied to the hidden states of the flow component across specific links to compute the delay, the jitter, and the packet loss.

Applications and model configurations. We consider the network modeling applications, listed in Table II that range across heterogeneous traffic and scheduling scenarios as well as prediction parameters such as delay, jitter, and packet-loss. The model configurations consist of the training/test data sources (`data_source`) that are listed in Table II, a range of epochs `epochs`$\in [1, 50]$, `steps_per_epoch`$\in [1, 2000]$, and the model deployment mode $\psi \in \{0, 1, 2\}$ that is defined in eqrefeq:mode. Overall, this results in a large number of possible configurations ($> 500k$) for each application.



(a) (b)

Fig. 4: (a) Inference-time/sample on compute nodes; and (b) Loss (MAPE) performance (config.: load-and-infer) for the applications listed in Table II.



(a) (b)

Fig. 5: (a) Training time/step on compute nodes; and (b) Loss (MAPE) (config.: 20 epochs, 2000 steps/epoch) for applications listed in Table II.

Compute Nodes. To evaluate the performance of the proposed QAG solution, we consider three types of compute nodes, CPU, GPU, and TPU [12], respectively, associated with the values of computational capability in trillions of operations per second (TOPS) and power consumption in watt (W), listed in Table III. We monitor the processor usage, architecture information, and power usage with the help of cross-platform libraries and commands (lscpu, psutil) along with the TensorFlow profiler and NVIDIA Management Library

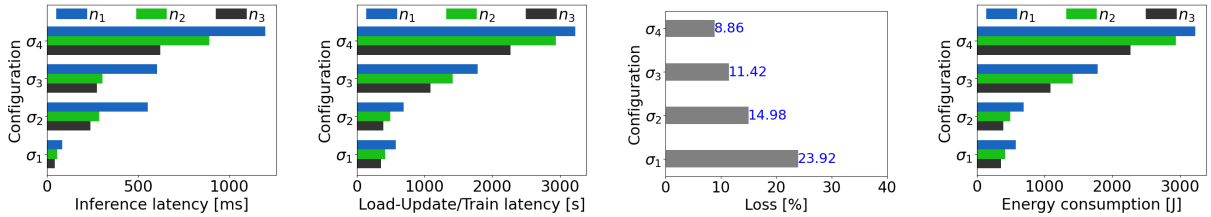


Fig. 6: Impact of the configurations (σ_{1-4}) execution by compute nodes (n_1:CPU, n_2:GPU, and n_3:TPU) listed in Table III: inference latency (left), load-update or train latency (center-left), loss (center-right) and energy consumption (right) for the h_1 application.

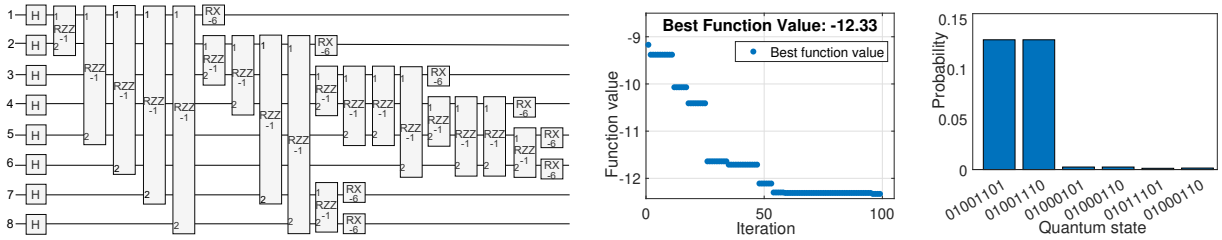


Fig. 7: Quantum circuit (left), iterative function value (center), and Quantum state probability (right) for the small-scale scenario shown in Fig. 2.

(nvml). We recall that such parameters define the computing resource capacity of the vertices $n_k \in \mathcal{V}_3$.

Benchmarks. We compare QAG against:

• *RouteNet-Fermi (RNF)*, [2]. We select [2] because it is a recent state-of-the-art for GNN-based network modeling for all the considered applications to predict delay, jitter, and packet-loss. However, it has a fixed configuration for model training and a fixed set of pre-trained models for inference.

• *Optimum (Opt),* obtained with brute-force exhaustive search.

B. Results and Discussions

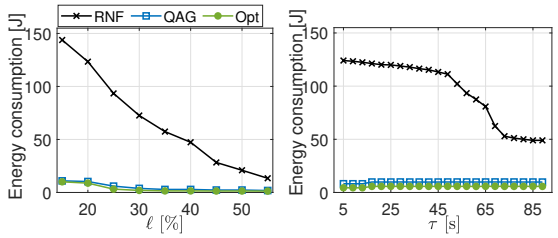
Latency, quality, and energy-consumption performance.

We evaluate the inference time per sample and loss performance (on average for >4000 samples) when using a pre-trained model without any updates/training in Fig. 4. We observe that the inference time per sample on the TPU is lesser than GPU, and it is lesser on GPU than on CPU. However, the quality of inference is very poor, i.e. inference loss is high ($> 100\%$ MAPE) for three applications (h_4, h_6, and h_7) while it is still considerable ($> 23\%$ MAPE) for the other applications. We note that even though inference using a pre-trained model is faster having at most 143 ms on TPU for h_4 application but it is not a viable option due to an extremely poor inference quality (extremely high loss). Hence, loading a pre-trained model for inference might not always be suitable to meet the network modeling application requirements.

Next, we evaluate the trade-off between training time (per step) and the quality of training configs. for the considered applications (Table II) and compute nodes (Table III) in Fig. 5. The loss performance, measured by MAPE, improves (i.e., loss decreases) with an increasing number of epochs for all the applications, but the extent of decrease in the loss is different for each application. Even the per-step training time is lesser on TPU as compared to GPU and CPU. Hence, an application and compute platform aware adaptive configuration selection and deployment is necessary for efficient network modeling.

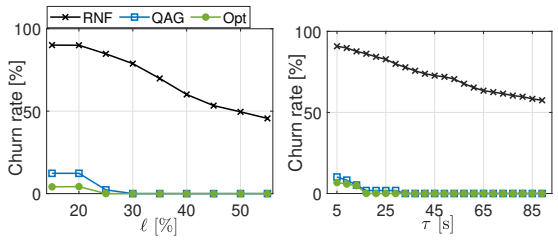
We further evaluate the performance (energy consumption, latency, and quality) of four sample configurations, $\sigma_1 = \{\text{ABILENE}, 1, 1, 0\}$, $\sigma_2 = \{\text{GEANT}, 1, 5, 1\}$, $\sigma_3 = \{\text{GEANT}, 10, 50, 1\}$, and $\sigma_4 = \{\text{GEANT}, 20, 200, 1\}$, that are deployed on compute nodes listed in Table III, for the application h_1 in Fig.6. This shows that a dynamic decision of inference or update-train after loading a pre-trained model on a chosen compute platform for a specific application has a direct impact on the performance (latency, quality, and energy).

Fig.7 shows the optimal quantum circuit, iterative objective function value, and the probability of the quantum state, for the small-scale example that is depicted in Fig. 2. The objective function, (6), converges to the minimum in less than 60 iterations. Specifically, we observe that the highest probability quantum states, $|01001101\rangle$ and $|01001110\rangle$, correspond to the sub-graphs $(h_1, \sigma_1, \sigma_2, n_{1 \text{ or } 2})$ and $(h_2, \sigma_3, \sigma_4, n_{2 \text{ or } 1})$.



(a) $\tau^h_{\max}$ =5 s (b) $\ell^h_{\max}$ =20%

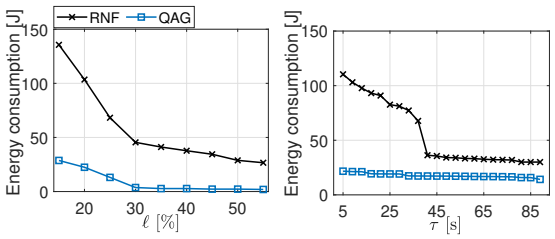
Fig. 8: Small scenario: Energy consumption as target loss and latency vary.



(a) $\tau^h_{\max}$ =5 s (b) $\ell^h_{\max}$ =20%

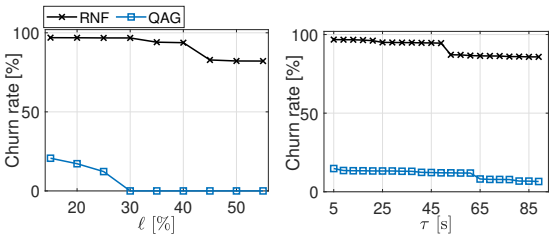
Fig. 9: Small scenario: Churn rate obtained as the target loss and latency vary.

Small-scale two-application scenario. We consider the small-scale two-application scenario similar to the example shown in Fig. 2 in order to evaluate the efficacy of our proposed



(a) $\tau^h_{\max}$ =5 s (b) $\ell^h_{\max}$ =20%

Fig. 10: Large scenario: Energy consumption as target loss and latency vary.



(a) $\tau^h_{\max}$ =5 s (b) $\ell^h_{\max}$ =20%

Fig. 11: Large scenario: Churn rate as the target loss and latency vary.

QAG solution with respect to the Optimal solution (*Opt*). It consists of two-applications, four configurations, and two compute nodes, that are uniformly randomly selected from the possible options for each. We evaluate the energy consumption and churn-rate (proportion of instances failing to meet the application requirements) performance over 10000 iterations with 95% confidence interval, with varying latency and loss requirements for the applications.

Fig.8(a) and Fig.8(b) show the energy-efficiency, Fig.9(a) and Fig.9(b) show the churn-rate performance, of the RNF, QAG, and Opt schemes in the small-scale scenario. We observe that both the energy consumption and the churn rate reduce as the latency target or the loss target increases due to the increased number of feasible configurations meeting the requirements. The proposed QAG performance closes matches the Opt, and it outperforms the RNF scheme in all considered requirement (loss and latency) settings. Overall, QAG results in at least a 70% lower energy consumption and a 60% lower churn rate than the benchmark RNF scheme.

Large-scale seven-application scenario. We consider a large scale scenario consisting of the seven applications listed in Table II, twenty possible configurations for each application, and three compute nodes of each type that are listed in Table III. The optimal solution is infeasible to obtain for a large scenario consisting of many applications, possible configurations, and compute nodes. This is due to the extremely high complexity of the exhaustive search method in finding the optimal solution for a large scenario that results in a large tripartite graph consisting of many vertices ($N > 10$). Hence, we perform the comparative energy consumption and churn-rate evaluation of the proposed QAG with respect to the state-of-the-art RNF scheme for varying latency and loss requirements of the considered applications.

For the large-scale scenario with loss target, $\ell^h_{max} = 20\%$, Fig.10(a) and Fig.11(a) show the energy-efficiency and churn-rate performance of the RNF and QAG schemes as the latency target varies. When we set the latency target, $\tau^h_{max} =$ 5s for the applications, Fig.10(b) and Fig.11(b) show the performance as the loss target varies. We observe that both the energy consumption and the churn rate reduces for QAG as the latency target or the loss target for the applications increase. This is due to the increased number of feasible configurations and need for lesser compute resources to meet the less stringent application requirements. The proposed QAG yields in atleast a 50% lower energy consumption and a 80% lower churn rate than the benchmark RNF scheme.

V. Conclusions

We solved the problem of orchestrating the adaptive GNN-based network modeling in the mobile-edge-cloud continuum system in an energy-efficient way. The network modeling applications using the GNN-based models can have diverse latency and inference quality (loss) requirements, we meet these by dynamically selecting the efficient GNN-based model configuration and deploying it on the suitable network compute node. Our solution approach, QAG, leverages the tripartite graph representation of the system and applies a low-complexity quantum approximation optimization to find the energy-efficient orchestration solution for heterogeneous network modeling applications. Extensive evaluation with different, co-existing network modeling applications for a small-scale and large-scale scenario demonstrates that our solution performs very closely to the optimum and, compared to the existing alternative, it can reduce the energy consumption by more than 50% while meeting the application requirements with at least a 60% lower churn-rate.

Acknowledgement

This work was supported by the 5G Events Labs project funded by BPI France and the PEPR 5G project funded by the French National Research Agency (ANR).

References

[1] H. S. Kuttivelil, S. Sreenivasamurthy, L. Krishnaswamy, *et al.*, "Network simulation bridge: Bridging applications to network simulators," in *Proc. ACM Intern. Symp. Q2SWinet*, 2023, pp. 39–46.

[2] M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, *et al.*, "RouteNet-Fermi: Network modeling with graph neural networks," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 3080–3095, 2023.

[3] S. Ding, J. Li, Y. Wu, *et al.*, "Transnet: A high-accuracy network delay prediction model via transformer and GNN in 6G," in *Proc. IEEE WCNC*, Apr. 2024.

[4] S. Huang, Y. Wei, L. Peng, *et al.*, "Xnet: Modeling network performance with graph neural networks," *IEEE/ACM Trans. Netw.*, vol. 32, pp. 1753–1767, 2024.

[5] C. Singhal, Y. Wu, F. Malandrino, *et al.*, "Resource-aware deployment of dynamic dnns over multi-tiered interconnected systems," in *Proc. IEEE INFOCOM*, 2024.

[6] C. Singhal, Y. Wu, F. Malandrino, *et al.*, "Resource-efficient sensor fusion via system-wide dynamic gated neural networks," in *Proc. IEEE SECON*, Dec. 2024.

[7] X. Li, K. Xie, X. Wang, *et al.*, "Tripartite graph aided tensor completion for sparse network measurement," *IEEE Trans. Parallel & Distributed Sys.*, vol. 34, no. 1, pp. 48–62, 2023.

[8] F. K. Hwang *et al.*, "Steiner tree problems," *Networks*, 1992.

[9] L. Zhou, S.-T. Wang, S. Choi, *et al.*, "Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices," *Phys. Rev. X*, vol. 10, p. 021 067, 2 Jun. 2020.

[10] A. J., A. Adedoyin, J. Ambrosiano, *et al.*, "Quantum algorithm implementations for beginners," *ACM Trans. Quantum Computing*, vol. 3, no. 4, Jul. 2022.

[11] X. Zou, R. Xie, Q. Tang, *et al.*, "Joint transmission and transcoding in computing power networks for livecast: A quantum-inspired optimization approach," in *Proc. IEEE WCNC*, Apr. 2024, pp. 1–6.

[12] Y. Wang, G. Wei, and D. Brooks, "Benchmarking tpu, gpu, and CPU platforms for deep learning," *CoRR*, vol. abs/1907.10701, 2019. arXiv: 1907.10701.