DeepGANTT: A Scalable Deep Learning Scheduler for Backscatter Networks

Daniel F. Perez-Ramirez
RISE Computer Science
Stockholm, Sweden
KTH Royal Institute of Technology
Stockholm, Sweden
daniel.perez@ri.se

Carlos Pérez-Penichet
RISE Computer Science
Stockholm, Sweden
carlos.penichet@ri.se

Nicolas Tsiftes
RISE Computer Science
Stockholm, Sweden
nicolas.tsiftes@ri.se

Thiemo Voigt
Uppsala University
Uppsala, Sweden
RISE Computer Science
Stockholm, Sweden
thiemo.voigt@angstrom.uu.se

Dejan Kostić
KTH Royal Institute of Technology
Stockholm, Sweden
RISE Computer Science
Stockholm, Sweden
dmk@kth.se

Magnus Boman
KTH Royal Institute of Technology
Stockholm, Sweden
mab@kth.se

ABSTRACT

Novel backscatter communication techniques enable battery-free sensor tags to interoperate with unmodified standard IoT devices, extending a sensor network's capabilities in a scalable manner. Without requiring additional dedicated infrastructure, the battery-free tags harvest energy from the environment, while the IoT devices provide them with the unmodulated carrier they need to communicate. A schedule coordinates the provision of carriers for the communications of battery-free devices with IoT nodes. Optimal carrier scheduling is an NP-hard problem that limits the scalability of network deployments. Thus, existing solutions waste energy and other valuable resources by scheduling the carriers suboptimally. We present DeepGANTT, a deep learning scheduler that leverages graph neural networks to efficiently provide near-optimal carrier scheduling. We train our scheduler with optimal schedules of relatively small networks obtained from a constraint optimization solver, achieving a performance within 3% of the optimum. Without the need to retrain, our scheduler generalizes to networks 6× larger in the number of nodes and 10× larger in the number of tags than those used for training. DeepGANTT breaks the scalability limitations of the optimal scheduler and reduces carrier utilization by up to 50% compared to the state-of-the-art heuristic. As a consequence, our scheduler efficiently reduces energy and spectrum utilization in backscatter networks.

CCS CONCEPTS

• **Networks** → **Sensor networks**; • **Computing methodologies** → **Machine learning**; **Planning and scheduling**.

IPSN '23, May 9–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0118-4/23/05.
https://doi.org/10.1145/3583120.3586957

KEYWORDS

scheduling, machine learning, wireless backscatter communications, combinatorial optimization

ACM Reference Format:
Daniel F. Perez-Ramirez, Carlos Pérez-Penichet, Nicolas Tsiftes, Thiemo Voigt, Dejan Kostić, and Magnus Boman. 2023. DeepGANTT: A Scalable Deep Learning Scheduler for Backscatter Networks. In *The 22nd International Conference on Information Processing in Sensor Networks (IPSN '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3583120.3586957

1 INTRODUCTION

Backscatter communications enable a new class of wireless devices that harvest energy from their environment to operate without batteries [18, 31, 53]. Recent advances have demonstrated how these battery-free backscatter devices—*tags* for short— can seamlessly perform bidirectional communications with unmodified Commercial Off-The-Shelf (COTS) wireless devices over standard physical layer protocols when assisted by an external unmodulated carrier [9, 22, 25, 26, 40, 43, 46]. This synergy facilitates new applications where tags are placed in hard-to-reach locations to perform sensing without the encumbrance of bulky batteries and the maintenance cost of frequent battery replacements [40, 43]. However, for backscatter tags to communicate, the Internet of Things (IoT) devices in the network must cooperate to provide an unmodulated Radio Frequency (RF) carrier. Unfortunately, providing carrier support means a significant resource investment for the IoT devices, which may be battery-powered. As a consequence, the efficient provision of unmodulated carriers is crucial for system-level performance and network lifetime.

Scenario. We consider a network of unmodified COTS wireless IoT nodes augmented with battery-free tags that do sensing on their behalf [40]. In this context, one can see the tags as devices that wirelessly provide additional functionality to the IoT nodes with simplicity akin to adding Bluetooth peripherals to our computers, but without incurring extensive maintenance and deployment costs associated with battery-powered devices [40, 43]. Since we aim at augmenting the sensing capabilities of unmodified standard

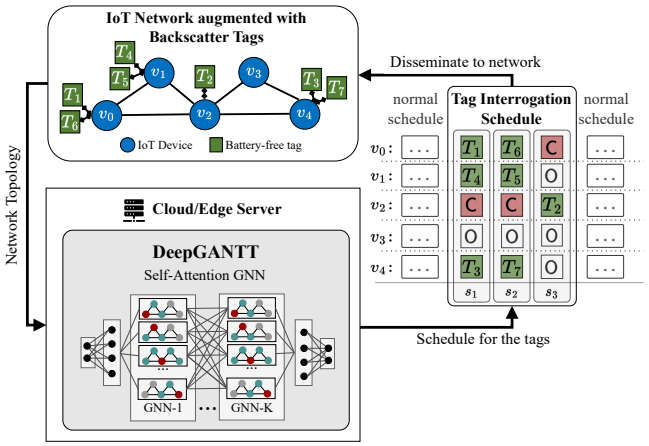


Figure 1: *DeepGANTT uses GNNs to schedule wireless communications in a backscatter network.* It takes a graph representing the wireless network as input and produces a schedule, which directs IoT nodes (v) to interrogate every battery-free tag (T) with minimal resources by reducing the number of carriers (C) and timeslots (s) needed.

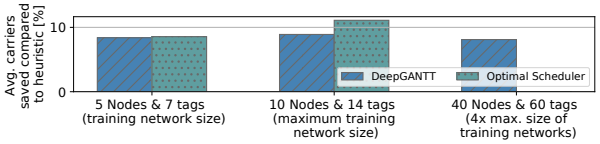


Figure 2: *DeepGANTT provides near-optimal schedules and generalizes well beyond the scalability limitation of the optimal scheduler.* DeepGANTT's performance is within 3% of the optimum on trained problem sizes, but can be applied to much larger networks.

wireless networks, we focus on scenarios where the IoT nodes employ a time-slotted MAC protocol. Such is the case of both Bluetooth Low Energy (BLE) and IEEE 802.15.4/ZigBee. The IoT nodes perform their own communication and computation according to their normal schedule. Additionally, a *tag interrogation schedule* is required for the commodity devices to collect sensor readings from the tags by coordinating among themselves to provide the unmodulated carrier that tags need to both receive and transmit data (see Fig. 1). An advantage of this *tag-augmented* scenario is that the battery-free tags can be located in hard-to-reach locations such as moving machinery, medical implants, or embedded in walls and floors. Meanwhile, the more capable IoT nodes are placed in accessible locations nearby, where either battery replacement or mains power is available [40, 43]. Consider, for instance, a healthcare monitoring application that includes implanted and wearable sensors [23]. If the battery-powered wearables cooperate to collect measurements from the implants, they could spare the patients from undergoing surgery just to replace the implants' batteries because they can now be battery-free. Making these devices battery-free is also important for sustainability reasons. Similar examples can be envisioned for applications such as industrial machinery, smart agriculture, and infrastructure monitoring [19, 47].

Challenges. In tag-augmented scenarios, IoT nodes invest considerable resources to enable the tags' communication, despite the fact that they may be battery-powered or otherwise resource-constrained. To minimize the resources allocated to supporting

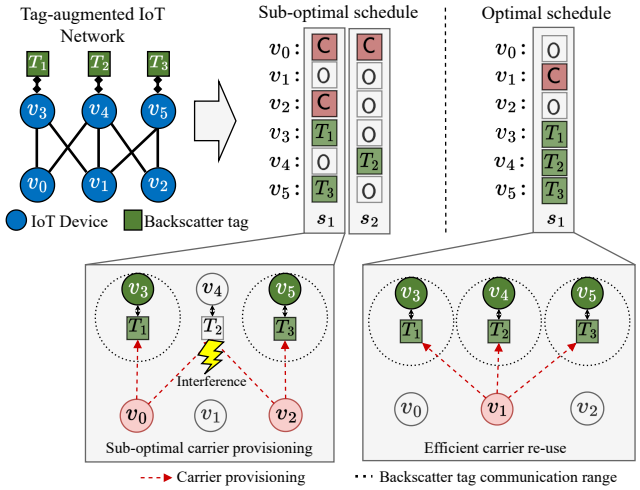


Figure 3: *Optimal schedules favor maximum carrier re-use.* Example of a tag-augmented IoT network and two possible schedules: the *optimal schedule* that maximizes carrier re-use while minimizing schedule length, and a *sub-optimal schedule*, which requires two timeslots. This is because the carriers from v_0 and v_2 interfere at T_2 in s_1, preventing it from being interrogated and forcing an additional timeslot for that purpose.

the tags, one must devise an efficient communication schedule for tag interrogations [41, 43]. A single tag interrogation corresponds to a request-response cycle between an IoT node and one of its hosted battery-free sensor tags, while exactly one of its neighboring IoT nodes provides an unmodulated carrier. More than one unmodulated carrier impinging on any tag causes interference and prevents proper interrogation. Provided that collisions are avoided, scheduling multiple tag interrogations concurrently reduces the schedule's length, improving latency and throughput in the network (see Fig. 3). Furthermore, scheduling one unmodulated carrier to serve multiple tags simultaneously greatly reduces the required energy and spectrum occupancy.

Computing the optimal schedule—minimizing the number of times nodes must generate carriers and minimizing the duration of the schedule—is an NP-hard combinatorial optimization problem [41]. At first glance, tag scheduling is similar to classic wireless link scheduling in that collisions among data transmissions must be avoided by considering the network topology. Carrier scheduling differs, however, in that we must additionally select appropriate carrier generators while avoiding collisions caused by them upon their neighbors. Simultaneously, we must also minimize resource utilization. The only known scalable solution is a carefully crafted heuristic, which suffers from suboptimal performance [41]. This results in wasted energy and spectral resources, particularly as network sizes grow. We refer to this heuristic as the *TagAlong scheduler* [43] hereafter. Alternatively, one can compute optimal solutions using a constraint optimization solver. We will refer to this scheduler as the *optimal scheduler*. This scheduler, however, takes a prohibitively long time as network sizes grow, which limits the capacity to adapt to network topology changes. For example, computing a schedule for a network of 10 nodes and 14 tags can take up to several hours.

In this paper, we leverage Deep Learning (DL) methods to overcome both the scalability limitations of the optimal scheduler and

the performance shortcomings of the TagAlong scheduler. However, the DL approach presents its own set of challenges. First, traditional DL methods that have succeeded on problems with fixed structure and input/output sizes (e.g., images and tabular data) are not applicable to the carrier scheduling problem since the latter operates on an irregular network structure, the size of which depends on the number of IoT nodes and sensor tags. Hence, using traditional DL methods would require training separate schedulers for different network sizes. Second, any given IoT network configuration may have many equivalent optimal solutions, which can confuse Machine Learning (ML) models during training. This is due to symmetries inherent to the carrier scheduling problem, i.e., the schedules are invariant to timeslot permutations.

Approach. In this work, we present *Deep Graph Attention-based Network Time Tables* (DeepGANTT), a new scheduler that builds upon recent advances in DL to efficiently schedule the communications of battery-free tags and the supporting carrier generation in a network of IoT devices interoperating with battery-free tags. DeepGANTT receives as input the network topology represented as a graph, and generates a corresponding interrogation schedule (Fig. 1). The graph representation of the network assumes there is a link between two IoT nodes if and only if there is sufficient signal strength between them for providing an unmodulated carrier. DeepGANTT iteratively performs one-shot node classification of the network's IoT devices to determine the role each of them will play within every schedule timeslot: either remain off (`O`), interrogate one of its tags (`T`), or generate a carrier (`C`), while also avoiding collisions in the network. The objective of the carrier scheduling problem is to reduce the resources needed to interrogate every tag in the network. By minimizing the number of carrier generation slots in the schedule, we reduce energy and spectrum occupancy. As a secondary objective, minimizing the number of required timeslots improves latency and throughput.

We adopt a supervised learning approach based on Graph Neural Networks (GNNs) instead of other paradigms such as reinforcement learning. This choice is mainly motivated by three facts. First, we can leverage the optimal scheduler to generate the training data necessary for a supervised approach. Second, GNNs are particularly successful in handling irregularly structured, variable-size input data such as network topology graphs [13, 28, 44]. GNNs provide a natural way of capturing the interdependence among neighboring nodes across multiple hops, crucial to avoid collisions. Third, it is straightforward to cast the scheduling problem as a classification task, which is generally tackled with a supervised approach [3, 35].

The use of GNNs allows us to train DeepGANTT only once using a set of network topologies, to then perform inference in other, previously unseen, network topologies of different size and structure. To train our scheduler, we generate random network topologies small enough for the optimal scheduler to handle. To avoid the pitfalls of training with multiple optimal solutions per instance, we add symmetry-breaking constraints to the optimal scheduler. Such constraints alter neither the true constraints, nor the objective of the problem. Instead, they narrow the choices of the solver from potentially many equivalent optimal solutions down to a single consistent one. For example, in tag scheduling, the order in which tags are interrogated is irrelevant. Nevertheless, by adopting a specific order (e.g., decreasing order of tag ID) we reduce the number of solutions from the factorial of the number of tags to one.

Contributions. We make the following specific contributions:

- We present DeepGANTT, the first fast and scalable DL scheduler that leverages GNNs to obtain near-optimal solutions to the carrier scheduling problem.
- We employ symmetry-breaking constraints to limit the solution space of the carrier scheduling problem when generating the training data using the optimal scheduler.
- DeepGANTT performs within 3% of the optimum in trained network sizes. Without the need to retrain, it scales to 6× larger networks while reducing carrier generation slots by up to 50% compared to the state-of-the-art heuristic. This directly translates to energy and spectrum savings.
- We use DeepGANTT to compute schedules for a real network topology of IoT devices. Compared to the heuristic, our scheduler reduces the energy per tag interrogation by 13% in average and up to 50% for large tag deployments.
- DeepGANTT's inference time is polynomial on the input size, achieving on average 429 ms and always below 1.5 s; a radical improvement over the optimal scheduler.

We show that DeepGANTT computes more resource-efficient schedules than the TagAlong scheduler, and that these are almost as good as those of the optimal scheduler (see Fig. 2). Moreover, DeepGANTT breaks the scalability limitations of the optimal scheduler by generating schedules for considerably larger backscatter networks while still reducing the energy consumption and spectrum occupancy compared to the heuristic. Finally, our scheduler's low inference times facilitates timely reactions to changes in topology and radio propagation conditions.

The rest of the paper is organized as follows. Sec. 2 gives background information that is useful to understand the paper. Sec. 3 formally describes the carrier scheduling problem. Sec. 4 details the design of DeepGANTT, while Sec. 5 discusses the implementation and training of the model. In Sec. 6, we evaluate DeepGANTT's performance against previous alternatives and prove that it can also compute schedules for a real IoT network. Sec. 7 discusses practical aspects and limitations of our scheduler. Lastly, we discuss related work in Sec. 8 and conclude our work in Sec. 9.

2 BACKGROUND

This section gives a quick overview of backscatter communications and the concept of *tag-augmented* IoT networks, with an intuitive introduction to GNNs.

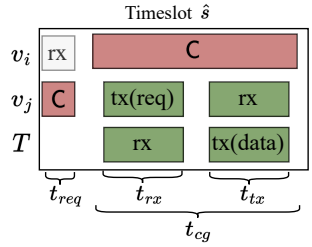
2.1 Backscatter Communications

Backscatter communication devices are highly attractive because their characteristic low power consumption enables them to operate without batteries. Instead, they can sustain themselves by collecting energy from their environment using various energy harvesting modalities. This kind of device achieves its low power consumption by offloading some of the most energy-intensive functions, such as the local oscillator, to an external device that provides an unmodulated carrier. Recent works have extended this principle to enable battery-free tags capable of direct two-way communications with unmodified COTS wireless devices using standard protocols such as

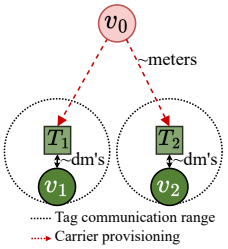
IEEE 802.15.4/ZigBee or Bluetooth, when supported by an external carrier [9, 22, 25, 26, 40, 43, 46]. Previous works have demonstrated systems that apply these battery-free communication techniques to augment an existing network of COTS wireless devices with battery-free tags [40, 41, 43]. We refer to this architecture as a *tag-augmented* IoT network. It enables placing sensors in hard-to-reach locations without having to worry about wired energy availability or battery maintenance.

We adopt a model where each tag is associated with (or hosted by) one IoT node responsible for interrogating it to collect sensor readings. Every IoT node in the network may host zero or more tags, and these are located in close proximity (within decimeters) of their host. The IoT nodes in the network are equipped with radio transceivers that support standard physical layer protocols such as IEEE 802.15.4/ZigBee or Bluetooth. They are able to provide an unmodulated carrier (by using their radio test mode [40]) and employ a time-slotted medium access mechanism. The use of a time-slotted access mechanism is motivated by its standardized use in commodity IoT devices [4, 21], and its simplicity for integrating the battery-free tags in the network. By design, a time-slotted access mechanism prevents channel collisions among tags hosted by an IoT node, as the node can communicate with only one of its tags in each timeslot (e.g., see how v_1 queries tags T_1 and T_6 in Fig. 1). The duration of a timeslot is sufficient for an IoT node to interrogate one tag by transmitting a request directed to the desired tag and receiving the response. Fig. 4a shows a tag interrogation procedure from an IoT node v_j when assisted by an unmodulated carrier from node v_i. First, the carrier providing node v_i listens for a period t_{req} for a request from the interrogating node v_j at its assigned timeslot in the schedule. Upon the request arrival, v_i provides a carrier for the duration t_{cg}. The interrogating node v_j transmits its request to one of its hosted tags T, after which the tag transmits its response back to v_j. As shown in Fig. 4b, the short communication range of the tags enables re-using an unmodulated carrier to perform multiple concurrent tag interrogations within a timeslot. However, for an IoT node to interrogate one of its hosted tags, exactly one of its neighboring IoT nodes must provide an unmodulated carrier [41, 43]. Multiple impinging carriers from neighboring nodes would cause interference on the tag, thus preventing proper communication between the tag and its host [43].

For these reasons, a schedule is required to coordinate how the IoT nodes cooperate among each other for providing the unmodulated carrier needed to interrogate their hosted tags. When not querying the backscatter tags, the network of IoT nodes performs its own computation and communication tasks according to its regular schedule. An interrogation schedule consists of one or more scheduling *timeslots*, each assigning one of three roles to every IoT node in the network: provide an unmodulated carrier (C), interrogate one of its hosted sensor tags (T), or remain idle (0). Existing studies have shown how the IoT nodes invest energy to provide carrier support in proportion to the number of carrier slots (C) scheduled, and that tags add latency proportionally to the length of the interrogation schedule [41]. As a consequence, it is critical that we optimize the way unmodulated carrier support is provided (see Fig. 3). For this reason, in this work, we focus on the efficiency of the scheduler, given that it bears total influence on resource expenditure. At least one of the IoT nodes in the network is connected to a cloud or edge



(a) *Tag uplink and downlink communication within a timeslot $\hat{s}$.*



(b) *Carrier re-use enabled by tags' short communication range.*

Figure 4: *The duration of a timeslot is sufficient for an IoT device to interrogate one tag by transmitting a request directed to the desired tag and receiving the response.* Furthermore, we leverage the short tag communication range to use one carrier provider to perform multiple tag interrogations across nodes in the network.

server where the interrogation schedule is computed. By keeping track of link state and node neighborhood information obtained from the physical layer and routing protocol, the IoT nodes can determine the connectivity graph among themselves. For purposes of tag interrogation, an edge between two IoT nodes in the connectivity graph represents a sufficiently strong wireless signal for carrier provisioning. This information can be relayed periodically or on demand to the cloud or edge server, where it is, together with the tag-to-host mapping, used to assemble the graph representation of the wireless network. Our scheduler uses this graph representation to produce a schedule, as depicted in Fig. 1.

2.2 Graph Neural Networks

GNNs have emerged as a flexible means to tackle various inference tasks on graphs, such as node classification [15, 44, 55]. Intuitively, stacking K GNN layers corresponds to generating node embedding vectors taking into account its K-hop neighborhood by leveraging the structure of the graph and inter-node dependencies [13, 28]. These embeddings are typically further processed with linear layers to produce the final output according to the task of interest. For example, one might perform node classification by passing each node embedding vector through a classification layer. Formally, given a graph $G = \langle V, E \rangle$ defined by the sets of nodes $v \in V$ and edges $(v, u) \in E$, at GNN layer i each node feature vector h_v is updated as:

$$h_v^{(i)} = f_1 \left(h_v^{(i-1)}, \underset{u \in \mathcal{N}(v)}{\text{AGG}} \left[f_2 \left(h_u^{(i-1)} \right) \right] \right), \tag{1}$$

where $\mathcal{N}(v)$ is the set of neighboring node feature vectors of node v, AGG is a commutative aggregation function, and f_1, f_2 are non-linear transformations [13]. Among the main advantages of using GNNs over traditional DL methods are their capability to

exploit the structural dependencies of the graph. Furthermore, they are an inductive reasoning method, i.e., GNNs can be deployed to perform inference on graphs other than those seen during training without the need to re-train the model [16, 49, 50].

3 PROBLEM FORMULATION

This section formally describes the carrier scheduling problem, i.e., efficiently scheduling the communications of sensor tags and the supporting carrier generation in a network of IoT nodes interoperating with battery-free tags. We hereon refer to the IoT devices and to the sensor tags in the network simply as *nodes* and *tags*, respectively. We model the wireless IoT network as an undirected connected graph G, defined by the tuple $G = \langle V_a, E \rangle$, where V_a is the set of N nodes in the network $V_a = \{v_i\}_{i=0}^{N-1}$, and E is the set of edges between the nodes $E = \{\langle u, v \rangle | u, v \in V_a\}$.

The connectivity among nodes in the graph (edges set E) is determined by the link state information collected as described in Section 2.1, i.e., there is an edge between two nodes if and only if there is a sufficiently strong wireless signal for providing the unmodulated carrier [41, 43]. We denote the set of T tags in the network as $N_T = \{T_m\}_{m=0}^{T-1}$, and their respective tag-to-host assignment as $H_T : T_m \in N_T \mapsto v_i \in V_a$. A node can host zero or more tags. The role of a node v_i within a timeslot $\hat{s}$ is indicated by the map $R_{vi,\hat{s}} : v_i \in V_a, \hat{s} \in [1, L] \mapsto \{\mathtt{C}, \mathtt{T}, \mathtt{0}\}$, where L is the schedule length in timeslots. Hence, a timeslot vector s_j is an N-dimensional vector containing the roles assigned to every node during the timeslot: $s_j = \left[R_{vi,j} | \forall v_i \in V_a\right]^\top$, so $j \hat{=} \hat{s} \in [1, L]$. A timeslot duration is long enough to complete one interrogation request-response cycle between a node and a tag (see Fig. 4a).

For a given problem instance (wireless network configuration) defined by the tuple $g = \langle G, N_T, H_T \rangle$, the Combinatorial Optimization Problem (COP) of interrogating all sensor values in the network once using the lowest number of carrier generators and timeslots is formulated as follows:

$$\min \quad (T \times C + L) \tag{2}$$

$$\text{s.t.} \quad \forall\, T_m \in N_T \quad \exists!\, \hat{s} \in [1, L] : R_{H_T,\hat{s}} = \mathtt{T} \tag{3}$$

$$\forall\, \hat{s} \in [1, L] \quad \forall T_m \in N_T \mid R_{H_T,\hat{s}} = \mathtt{T}$$

$$\exists!\, v_i \in V_a : R_{vi,\hat{s}} = \mathtt{C} \wedge (H_T, v_i) \in E\ , \tag{4}$$

where C is the total number of carriers required in the schedule, i.e., $C = \left|\{R_{vi,\hat{s}} = \mathtt{C} : v_i \in V_a, \hat{s} \in [1, L]\}\right|$. Constraint (3) enforces that tags are interrogated exactly once in the schedule. Additionally, constraint (4) prevents channel collisions among carriers by ensuring that there is only one carrier-providing neighbor per interrogated tag in each timeslot. The objective function (2) is designed to prioritize reducing the number of carrier slots (C) over the duration of the schedule (L). This is because we are most concerned with energy and spectrum efficiency and because a reduction of C often implies a reduction of L, but the converse is not necessarily true. For example, in Fig. 1, v_2 provides a carrier to interrogate T_1 and T_3, reducing C and L simultaneously.

4 SYSTEM DESIGN

In this section, we first introduce the design considerations for DeepGANTT to cope with the wireless network requirements and the challenges in carrier scheduling. We then introduce DeepGANTT's system architecture.

4.1 Design Considerations

DeepGANTT is deployed at an edge or cloud server, where schedules are computed on demand for the IoT network. At least one of the IoT nodes in the network is assumed to be connected to the Edge/Cloud server, and it is responsible for building the IoT network topology graph, emitting the request to the scheduler in the Edge/Cloud, and disseminate the computed schedule to the other devices. The DeepGANTT scheduler receives as input the IoT network configuration as the tuple $g = \langle G, N_T, H_T \rangle$, i.e., the wireless network topology G and the set of tags N_T in the network with their respective tag-to-host assignment H_T. The scheduler then generates the interrogation schedule $S = [s_j]_{j=1}^{L}$ and delivers it to the IoT network. The scheduler may receive subsequent schedule requests by the IoT network either upon addition/removal of nodes or tags, or upon connectivity changes among the IoT nodes. Thus, DeepGANTT must be able to react fast to structural and connectivity changes in the wireless network.

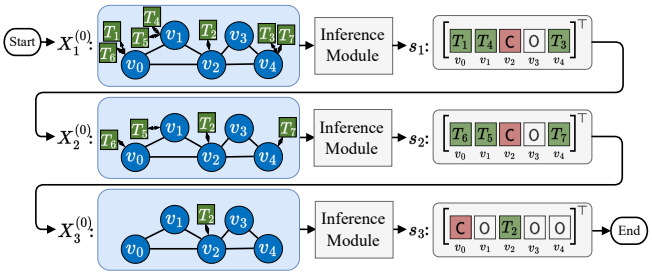
From an ML perspective, every possible configuration of g yields a different graph representation with different connectivity and potentially different input size. Likewise, the output for g (schedule S^g) may vary in size in terms of both the number of timeslots L and the number of nodes N in the topology (since $s_j \in \mathbb{R}^N \forall j \in [1, L]$). Moreover, at each timeslot, every node is assigned to one of three possible actions $\{\mathtt{C}, \mathtt{T}, \mathtt{0}\}$ (see Section 2.1) based on its neighborhood.

We use a GNN-based learning approach to allow DeepGANTT to process variable-sized inputs (network topology) and output (interrogation schedule) sequences while learning the local dependencies of a node in the network (see Sec. 2.2). For carrier scheduling, exploiting the structural dependencies of the nodes' K-hop neighborhood allows to efficiently schedule carriers while avoiding interference. Moreover, the use of GNNs allow us to train the ML model only once, and then deploy it without the need to retrain for previously unseen network configurations in terms of the number of nodes N, their connectivity, and the number of sensor tags T.

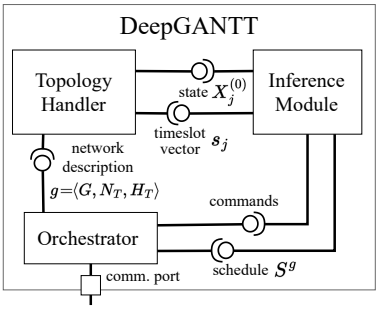
4.2 System Architecture

We model the carrier scheduling problem as an iterative one-shot node classification problem. The inner workings of DeepGANTT are illustrated in Fig. 5a: on each iteration j, DeepGANTT's inference module assigns each node in the topology to one of three classes corresponding to the possible node actions: $\{\mathtt{C}, \mathtt{T}, \mathtt{0}\}$. Hence, each iteration j generates a timeslot vector s_j. Each predicted timeslot is checked for compliance with the constraints in Eq. 3 and 4. After each iteration, the topology's node feature matrix $X_j^{(0)}$ is updated by removing one tag from the nodes that were assigned class $\mathtt{T}$ (interrogate). This process is repeated until all tags have been removed from the cached network configuration.

The DeepGANTT scheduler consists of three submodules, as depicted in Fig. 5b. The *Orchestrator* is DeepGANTT's coordinating unit and interacts with the outside world through its communication port. It receives the necessary information from the IoT network, including the network's link and routing layer information to construct the topology graph G. The *Orchestrator* is also



(a) *DeepGANTT iteratively performs one-shot node classification, one timeslot at a time, removing scheduled tags from the topology and repeating the process until no more tags remain.*



(b) *The three core components of DeepGANTT interact among themselves and with the exterior to generate schedules.*

Figure 5: *DeepGANTT's system architecture and the Inference Module's procedure to generate schedules.*

responsible for providing the problem description to the *Topology Handler* and interfacing with the *Inference Module*. The *Topology Handler* maintains the problem description, provides it to the *Inference Module* in a format suitable for the ML model, and updates its state according to the predicted timeslots. Since the *Inference Module* is trained on the basis of a stochastic process, the *Topology Handler* includes a fail-safe functionality to make sure that the predictions comply with the constraints in Eqs. (3) and (4). In case of failure, the *Topology Handler* restores compliance by randomly shuffling tag and node IDs and retrying. This does not alter the final schedules.

4.3 Input Node Features

At timeslot j, the Inference Module receives as input a node feature matrix $X_j^{(0)} \in \mathbb{R}^{N\times D}$ containing the row-ordered node feature vectors $x_{i,j}^{(0)} \in \mathbb{R}^D, \forall i \in V_a$, where D represents the number of features representing a node's input state. Since the tags lie in the close proximity of a node, and each of them interacts only with their host, we model them as a feature in their host's input feature vector. One can also include additional features to assist the GNN during inference. Hence, we consider three different node features:

- Hosted-Tags: the number of tags hosted by a node.
- Node-ID: integer identifying a node in the graph.
- Min. Tag-ID: the minimum tag ID among tags hosted by a node. Since a node can host several tags, the min. Tag-ID represents only the lowest ID value among its hosted tags.

Intuitively, the number of tags hosted by a node is decisive for assigning carrier-generating nodes. For example, if one node hosts all tags, this node should never provide an unmodulated carrier in the schedule. Similarly, the node hosting the greatest number of tags is unlikely to be a carrier provider in the schedule. For this reason, *Hosted-Tags* is always included as an input node feature. Moreover, including the node-ID and the minimum tag-ID can provide the scheduler with context on how to prioritize carrier-provider nodes, and with an order to interrogate the tags.

4.4 Inference Module

The Inference Module is the learning component of DeepGANTT and contains the ML model for performing inference. In the following, we describe in detail the Inference Module's ML architectural components depicted in Fig. 6.

Embedding. The input node feature matrix $X_j^{(0)}$ is first transformed by an *Embedding* component with the aim to assist the subsequent message-passing operations in performing better injective neighborhood aggregation. That is, enabling each node to better distinguish each of its neighbors' contributions. The embedding transformation of $X_j^{(0)}$ is described by:

$$H_j^{(0)} = \mathrm{LN}\left(\Big|\Big|\left[X_j^{(0)}, \mathrm{FNN}_e\left(X_j^{(0)}\right)\right]\right), \tag{5}$$

where LN corresponds to the layer normalization operation from Ba et al. [1], and $||$ represents the concatenation operation. FNN_e corresponds to a fully-connected neural network layer (FCNN) with a non-linear transformation as:

$$\mathrm{FNN}_e(X_j^{(0)}) = \mathrm{LeLU}\left(\mathrm{FCNN}_e(X_j^{(0)})\right) \tag{6}$$

$$\mathrm{FCNN}_e(X_j^{(0)}) = X_j^{(0)} W_e + b_e\ , \tag{7}$$

with the leaky ReLU operator LeLU and NN parameters $W_e \in \mathbb{R}^{D\times D_e}$, $b_e \in \mathbb{R}^{D_e}$. The embedding outputs the node feature matrix $H_j^{(0)} \in \mathbb{R}^{N\times(D+D_e)}$.

Stacked GNN Blocks. The output from the embedding component $H_j^{(0)}$ represents the input to a stack of K GNN blocks $\{B^{(i+1)}\}_{i=0}^{K-1}$. The inner structure of each GNN block is depicted in the lower part of Fig. 6. We stack K different GNN blocks to allow nodes to receive input from their K-hop transformed neighborhood. Moreover, we select summation as the aggregation operation (see Eq. 11) motivated by the results of Hamilton et al. [16]. Each GNN block $B^{(i+1)}$ receives as input the previous layer output $H_j^{(i)}$ and performs the following operations:

$$H_j^{(i+1)} = \mathbf{B}^{(i+1)}(H_j^{(i)}) \tag{8}$$

$$\mathbf{B}^{(i+1)}(H_j^{(i)}) = \mathrm{LN}\left(\tilde{H}_j^{(i+1)} + \mathrm{FNN}_B^{(i+1)}\left(\tilde{H}_j^{(i+1)}\right)\right) \tag{9}$$

$$\tilde{H}_j^{(i+1)} = \mathrm{LN}\left(H_j^{(i)} + \mathbf{G}^{(i+1)}(H_j^{(i)})\right). \tag{10}$$

Eq. 10 corresponds to a multi-head self-attention GNN activation $\mathbf{G}^{(i+1)}$ followed by an Addition&Normalization component in Fig. 6. Subsequently, Eq. 10 implements a per-node FCNN with a non-linear transformation, followed by another Add&Norm component. Analogous to Eq. 7, each block's per-node $\mathrm{FNN}_B^{(i+1)}$ has parameters $W_b^{(i+1)}$ and $b_b^{(i+1)}$ followed by a leaky-ReLU.

The operation $\mathbf{G}^{(i+1)}$ in Eq. 10 implements a scaled dot-product multi-head self-attention GNN [45, 48]. Each of the N row-ordered node feature vectors $H_j^{(i)} = \{h_{j,t}^{(i)}\}_{t=0}^{N-1}$ is updated at GNN layer $\mathbf{G}^{(i+1)}$ by the following message-passing operation (for simplicity, we omit the timeslot subscript j):

$$h_t^{(i+1)} = \text{FCNN}_{upt}^{(i+1)}\left(h_t^{(i)}\right) + \Big\|_{m=1}^{M}\left[\sum_{p\in\mathcal{N}(t)} \alpha_{m,tp}^{(i+1)}\tilde{v}_{m,p}^{(i+1)}\right] \quad (11)$$

$$\tilde{v}_{m,t}^{(i+1)} = \text{FCNN}_{m,val}^{(i+1)}\left(h_t^{(i)}\right) \quad (12)$$

$$\alpha_{m,tp}^{(i+1)} = \frac{\Gamma\left(q_{m,t}^{(i+1)}, k_{m,p}^{(i+1)}\right)}{\sum_{u\in\mathcal{N}(t)}\Gamma\left(q_{m,t}^{(i+1)}, k_{m,u}^{(i+1)}\right)} \quad (13)$$

$$q_{m,t}^{(i+1)} = \text{FCNN}_{m,qry}^{(i+1)}\left(h_t^{(i)}\right) \quad (14)$$

$$k_{m,t}^{(i+1)} = \text{FCNN}_{m,key}^{(i+1)}\left(h_t^{(i)}\right) \quad (15)$$

$$\Gamma(A,B) = \frac{\exp\left(A^\top B\right)}{\sqrt{d}},\ A,B\in\mathbb{R}^d\ . \quad (16)$$

Eq. 11 is the main node update operation for each node, consisting of adding two parts. First, a transformation of the target node's feature vector through $\text{FCNN}_{upt}^{(i+1)}$. Second, the concatenated M-heads of the attention operations with the target node's neighbors. Each self-attention operation m consists of a weighted sum over transformed neighboring node feature vectors $\tilde{v}_{m,t}^{(i+1)}$, inspired by the attention operations by Vaswani et al. [48]. Each of the neighbors' node feature vectors are transformed by the layer $\text{FCNN}_{m,val}^{(i+1)}$ in Eq. 12. The weights $\alpha_{m,tp}^{(i+1)}$ at attention head m from neighbor source p to target update node t are obtained through Eq. 13 over the scaled dot product operation Γ (Eq. 16) between query $q_{m,t}^{(i+1)}$ and key $k_{m,t}^{(i+1)}$ transformations of the target node (see Eq. 14) and the source neighbor node (see Eq. 15), respectively. There are multiple reasons motivating the use of attention. First, it allows to leverage neighboring nodes' contributions differently. Second, attention has provided better results for node classification tasks [49] compared to isotropic GNNs [16, 28]. Moreover, conventional GNN node classification benchmarks assume that nodes in a spatial locality of the graph assume similar labels and leverage the fixed-point theorem property in stacking GNNs [28]. This greatly contrasts with the carrier scheduling problem, in which neighboring nodes are mostly expected to have different classes (a node provides a carrier, neighbors interrogate tags). Finally, attention serves as a counteracting factor for this property when combined with skip-connections: it builds deeper networks that can learn contrasting representations at each layer.

Classification Layer. The output from the K GNN blocks is then fed to a FCNN followed by a Softmax for obtaining a per-node probability distribution over the classes that correspond to the possible node actions $\{\mathtt{C}, \mathtt{T}, \mathtt{0}\}$. Finally, each node is assigned to the class with the highest probability.

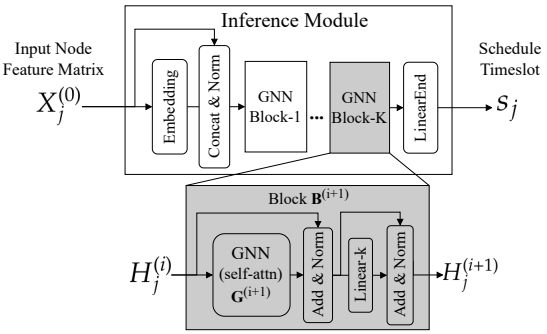


Figure 6: *The Inference Module implements DeepGANTT's ML model. At its core, there are K self-attention GNN blocks.*

Note that all parameters from the Inference Module listed (Eqs. 5–15) are shared across timeslots s_j.

5 LEARNING TO SCHEDULE

This section explores how the Inference Module's ability to compute interrogation schedules is influenced by two factors. On one hand, the interdependence between the choice of input node features and on the other the configuration of the training data generation. Additionally, we explore the influence of ML model complexity in terms of the number of GNN blocks on the Inference Module's performance. This exploratory analysis resulted in selecting an Inference Module composed of 12 GNN blocks.

5.1 Training Data Generation

The carrier scheduling problem, as described in Eqs. 2-4, presents symmetries that result in multiple optimal solutions which confuse the DL model while training. In the following, we describe how we leverage symmetry-breaking constraints in the constraint optimizer to generate a training dataset with unique optimal solutions.

As an example of these symmetries consider that: because the order of the timeslots in the schedule is irrelevant, a schedule of duration L is equivalent to $L!$ other schedules. A similar set of symmetries appears among all tags hosted by the same node as the order of interrogating them is also irrelevant. To make matters worse, a given set of tags can often be served by more than one carrier generator, therefore introducing more symmetries.

Symmetry-Breaking Constraints. To overcome the confusion that these symmetries might cause to a learning-based model during training, we further constrain the carrier scheduling problem in a way that eliminates symmetries but does not otherwise alter the problem. To that end, we enforce lexicographical minimization of a vector of length T that indicates the timeslot where each tag is scheduled. This automatically eliminates symmetries related to the order of tag interrogations. Similarly, we also lexicographically minimize another length-T vector containing the node that provides the carrier for each tag; which eliminates symmetries related to multiple potential carrier nodes.

Table 1: *The interrelation between data generation and input node features strongly impact the model's performance. Including symmetry-breaking constraints dramatically improves the model performance.* Interdependence of data generation and input feature configuration for an Inference Module of six GNN-Blocks. S_{corr} indicates percentage of problem instances for which the NN delivers a complete schedule. F1-score is provided for C (carrier class).

Symmetry breaking:	Disabled			Enabled		
Performance Metric [%]:	Accuracy	F1-score	S_{corr}	Accuracy	F1-score	S_{corr}
Features-1 (Hosted-Tags):	85.69	57.78	27.04	86.61	60.54	10.11
Features-2 (Hosted-Tags + Node-ID + Min. Tag-ID.):	86.40	59.82	47.96	**99.22**	**97.36**	**99.64**

5.2 ML Model Validation Setup

We implement all components in the Inference Module using PyTorch [39]. For the GNNs layers $\mathbf{G}^{(i+1)}$, we use the PyG self-attention based GNN *TransformerConv* implementation [11].

Input Features. We consider two different feature configurations for the input node feature matrix $X_1^{(0)} \in \mathbb{R}^{N \times D}$: *Features-1* includes only the *Hosted-Tags* ($D = 1$), and *Features-2* considers Features-1 plus the *Node-ID* and *minimum Tag-ID* among the tags hosted by a node ($D = 3$).

Validation Dataset. We generate small-sized problem instances of varying sizes and number of tags from two to ten nodes ($N \in [2, 10]$), and hosting one to 14 tags ($T \in [1, 14]$). We generate these graphs using the *random geometric* graph generator from NetworkX [14] to guarantee that these graphs can exist in 3D space. As the network size varies, we make sure to maintain a constant network spatial density. We uniformly assign tags to hosts at random. We employ a constraint optimizer to compute solutions for a total of 520000 problem instances both including and without including the symmetry-breaking constraints. As constraint optimizers, we employ MiniZinc [36] and OR-Tools [42]. The problem instances are divided into a train-set and a validation-set using a 80%-20% split. Since we perform a per-node classification for every scheduling timeslot (see Fig. 5a), we further consider each timeslot input-target pair $(X_j^{(0)}, s_j)$ as a training sample, which yields an approximate total of 1.5 million samples.

Validation Metrics. We consider both ML metrics and an application related metric to evaluate a model's performance. From the ML perspective, we employ overall accuracy, and the carrier class (C) F1-score due to its crucial role in avoiding signal interference for tag interrogation. For the application-related metric we consider the percentage of correctly computed schedules S_{corr}. This metric indicates the percentage of problem instances for which the already-trained ML model produces a complete (all timeslots) and correct (fulfilling carrier scheduling constraints) schedule. At inference time, the *Topology Handler* handles these unlikely cases as described in Section 4.2.

5.3 ML Model Training

We train the Inference Module with the Adam optimizer [27] on the basis of mini-batch gradient descent with standard optimizer parameters and an initial learning rate of 10^{-3}. The ML model should give greater importance to the carrier-generating node class (C) since it can subsequently determine the tags that can be interrogated or not due to signal interference. Hence, we build upon the cross-entropy loss for classification and propose an additional factor to account for greater importance to the carrier class (C) based on the L1 norm between the prediction and the ground-truth:

$$L_{batch} = \frac{1}{N_b} \sum_{i=0}^{N_b-1} \left(\left(- \sum_{k \in \{\mathtt{C},\mathtt{T}/\mathtt{O}\}} y_{ik} \log(\hat{y}_{ik}) \right) \times e^{||(\hat{y}_i == \mathtt{C}) - (y_i == \mathtt{C})||_1} \right) + \rho ||\hat{W}||_2^2 \quad (17)$$

where N_b is the number of nodes in the mini-batch, y_i and $\hat{y}_i$ are the ground-truth and model prediction of mini-batch sample i, respectively, ρ is the regularization hyperparameter, and $\hat{W}$ represents a tensor containing all the learning parameters undergoing gradient descent. We implement untunned warmup as presented by Ma & Yarats [32], followed by learning rate decay by 2% every epoch, and early stopping after 25 subsequent epochs without minimization of the test loss. We save the best-performing model on the basis of the F1-score.

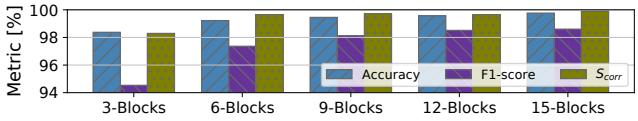


Figure 7: *Increasing the number of GNN-Blocks improves the performance of the NN model.* The NN reaches a saturation point for the F1-score at approx. 12 layers.

5.4 Validation Results

The Inference Module considered for evaluating data generation strategies and node feature representation consists of six GNN-blocks. We empirically chose the number of GNN-blocks for the first experiments based on a trade-off between model training time and performance. After establishing the best combination of input node feature representation and data generation strategy, we analyze the influence in performance of ML model complexity.

5.4.1 Data Generation and Input Features Interdependence. The influence of both input node feature representation and data generation configuration in model performance is depicted in Table 1. Regardless of the data-generating configuration, an enriched node feature representation (including more node features) leads to an increase in accuracy and F1-score. There are two possible reasons for this. First, by increasing the node feature dimension D, we ensure that the GNNs can perform more efficient injective neighborhood aggregation (to better distinguish neighboring node contributions). This is realized by making it less likely for two nodes to have the same input feature vector to the GNN. Second, both the Nodes-ID and Tags-ID feature serve as a node's positional encoding information in a graph, a property that assists the model

in breaking graph symmetries [8, 57]. Additionally, an enriched node feature representation leads to an increase of S_{corr}, regardless of the chosen data generation configuration: 20.92% and 89.53% improvement from Features-1 to Features-2 for the standard and the symmetry-breaking configurations, respectively. The constraint optimizer explicitly uses the Node-ID and Tag-ID to compute the optimal solution in the symmetry-breaking configuration, which is why providing the ML model with these features is crucial for it to be able to learn the structural dependencies in the topologies. Implementing symmetry-breaking measurements in the data generation procedure is a critical measure for allowing the ML model to generate complete schedules (highest increase in S_{corr}). Since we explore a supervised learning approach, it is crucial to constrain the mapping between inputs and targets for the NN model to learn consistent graph-related structural dependencies.

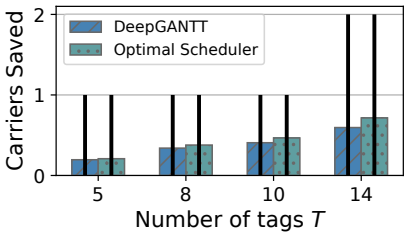
5.4.2 Influence of the Number of GNN-Blocks. To analyze the influence of the required K-hop neighborhood aggregation of the GNN model, we analyze the model's performance as we vary the number of GNN-Blocks while keeping other architectural components constant to the values found by extensive empirical analysis. Specifically, we set the embedding dimension to $D_e = 48$, the number of attention heads to $M = 2$, and the GNN-Blocks' hidden feature dimensions to $D_H^{(i)} = 200$. Fig. 7 illustrates how increasing the number of GNN-Blocks increases the performance of the ML model: the F1-score increases to a saturation point around 12 layers. Although the 15-layer model exhibits the highest S_{corr}, subsequent experiments showed that this model overfits and hence is unable to scale to larger problem instances.

Inference Module. Based the findings of this section, DeepGANTT's Inference Module consists of 12 GNN blocks and is trained on ~424000 small-sized problem instances (IoT networks of up to 10 nodes and 14 tags) obtained from the optimal scheduler as described in Section 5.3. We used an NVIDIA Titan RTX for 131 epochs before reaching the early-stop condition. The model achieved 99.56% accuracy, a carrier-class F1-score of 98.51%, and a percentage of correctly computed schedules of $S_{corr} = 99.88\%$.

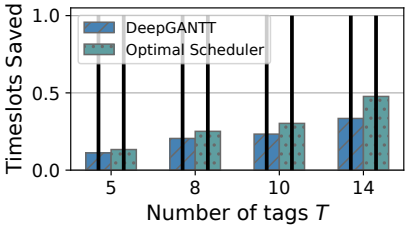
6 EVALUATION

After designing and training DeepGANTT's Inference Module in Section 5, in this section, we compare DeepGANTT's results to those of the TagAlong scheduler [41] and the optimal scheduler. By design, DeepGANTT can generate schedules for previously unseen topologies without the need for retraining. Hence, no further ML model training was performed for the experiments presented in this section. We highlight the following key findings:

- DeepGANTT performs within 3% of the optimal scheduler on the average number of carriers used, while consistently outperforming TagAlong by up to 50%. This directly translates into energy and spectrum savings for the IoT network.
- Our scheduler scales far beyond the problem sizes where it was trained, while still outperforming TagAlong; therefore enabling large resource savings well beyond the limits of the optimal scheduler.
- We deploy DeepGANTT to compute schedules for a real IoT network of 24 nodes. Compared to the TagAlong scheduler, DeepGANTT reduces the energy per tag interrogation by 13.1% in average and up to 51.6%.
- With polynomial time complexity and a maximum observed computation time of 1.49 s, DeepGANTT's speed is comparable to TagAlong's, and well within the needs of a practical deployment.



(a) *Average number of carriers saved relative to TagAlong.*



(b) *Average timeslot savings relative to TagAlong.*

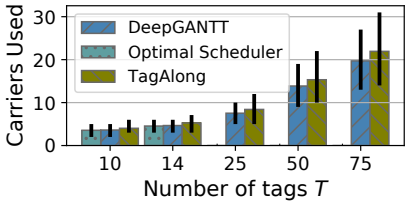
Figure 8: *DeepGANTT largely mimics the optimal scheduler both in terms of carriers saved and timeslots saved.* Average gain for networks of 10 IoT nodes and various numbers of tags in the test dataset. The black bars depict the 10 and 90 percentiles.

Baselines. To conduct our evaluation, we consider two baselines: i) the *optimal scheduler*, which corresponds to using a constraint optimizer for computing the optimal solution including the symmetry-breaking constraints for topologies of up to 10 nodes and 14 tags, and ii) the *TagAlong scheduler*, the state-of-the-art heuristic algorithm for computing interrogation schedules for the type of backscatter IoT network considered in this work [41].

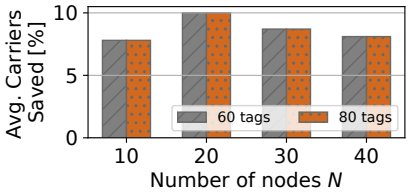
Evaluation Section Structure. This section is structured as follows. Sec. 6.1 benchmarks DeepGANTT's performance against the optimal scheduler and the TagAlong scheduler for topologies of up to 10 nodes and 14 tags previously unseen to DeepGANTT (*test* set). Sec. 6.2 analyzes DeepGANTT's gains over TagAlong for topologies well beyond the practical applicability of the optimal scheduler of up to 60 nodes and 160 tags (*generalization* set). Sec. 6.3 describes DeepGANTT's time complexity and its computation times. Finally, Sec. 6.4 demonstrates DeepGANTT's ability to produce schedules for a real IoT network of $N = 24$ nodes under different number of tags configurations.

6.1 Test Set Performance

We hereby demonstrate DeepGANTT's ability to mimic the behaviour of the optimal scheduler to produce interrogation schedules and exhibit similar gains as the optimal scheduler's performance over the TagAlong heuristic.



(a) *Average number of carriers used for networks of 10 IoT nodes.*



(b) *Average percentage of carriers saved compared to TagAlong.*

Figure 9: *DeepGANTT scales far beyond the range of data where the optimal schedules used in training are available, while outperforming TagAlong by a growing margin (9a)* The black bars depict the 10 and 90 percentiles. DeepGANTT also maintains an average saving of almost 10% in scheduled carriers while scaling up to four times the maximum training network size (9b).

Test Dataset. We consider topologies consisting of ∼106000 problem instances of up to 10 nodes and 14 tags for which it is still possible to deploy the optimal scheduler.

Test Metrics. We compare the number of carrier slots in the generated schedules since it directly affects the IoT network's overall energy consumption. Specifically, for every evaluation problem instance we compute DeepGANTT's saved carriers as $C_d - C_t$ and, those of the optimal scheduler as $C_o - C_t$; where C_d, C_t, and C_o are the number of carriers scheduled by the DeepGANTT, TagAlong and the optimal schedulers respectively. Furthermore, we analyze the total schedule length, since it directly relates to the latency of communications in the wireless network. We compute the timeslots savings in a manner analogous to the carrier savings.

Test Results. Fig. 8a shows the average number of carriers saved compared to TagAlong (higher is better) for various network sizes in the test dataset. DeepGANTT's performance is very close to that of the optimal scheduler in all cases. The number of timeslots in the schedule is the secondary objective in the tag scheduling problem; this is because, while the duration of the schedule can impact latency and other performance metrics in the network, the number of carriers directly impacts energy and spectral efficiency. Note that reducing the number of carrier slots, can potentially also shorten the schedule owing to carrier reuse [41]. As depicted in Fig. 8b, DeepGANTT largely mimics the performance of the optimal scheduler regarding timeslot savings. Our scheduler outperforms the TagAlong scheduler in 98.2% of the test-set instances. On those instances where TagAlong schedules are shorter, it is by one timeslot at most.

6.2 Generalization Performance

We now analyze the capabilities of DeepGANTT in computing schedules for problem sizes well beyond those observed during training and compare its performance against the TagAlong scheduler. The ability to generalize this way directly translates into better scalability than that of the optimal scheduler.

Generalization Dataset. We consider 1000 problem instances for every $\langle N, T \rangle$ pairs from the sets $N \in \{10, 20, 30, 40, 60\}$ and $T \in \{20, 40, 60, 80, 160\}$, i.e., 25000 different IoT wireless networks.

Metrics. We consider the same metrics as those used in Sec. 6.1.

Generalization Results. Fig. 9a shows a comparison of the average number of carriers utilized (lower is better) on problem sizes beyond those used in training. DeepGANTT outperforms TagAlong by a growing margin well beyond the maximum size of optimal solutions seen in training (beyond 10 nodes and 14 tags). Compared to TagAlong, DeepGANTT is able to reduce the percentage of necessary carriers $\sim 7\% - 10\%$ on average for large numbers of tags, as depicted in Fig. 9b. Fig. 10 depicts the number of carriers saved and the percentage of correctly computed schedules S_{corr} for different network size configurations. DeepGANTT consistently increases the mean number of carriers saved as the number of tags increases for all considered configurations. While there are cases where TagAlong outperforms DeepGANTT, these are actually rare occurrences; this is evident by the positive mean and the location of the 25-percentiles. Additionally, DeepGANTT's percentage of correctly-computed schedules (S_{corr}) decreases for the 10-node 80-tags case, but remains above 99% for the 40 and 60 nodes problem instances, for all the number of tags considered. DeepGANTT is able to reduce the number of carriers by up to 50% for all number of nodes configurations in Fig. 10.

6.3 Computation Time

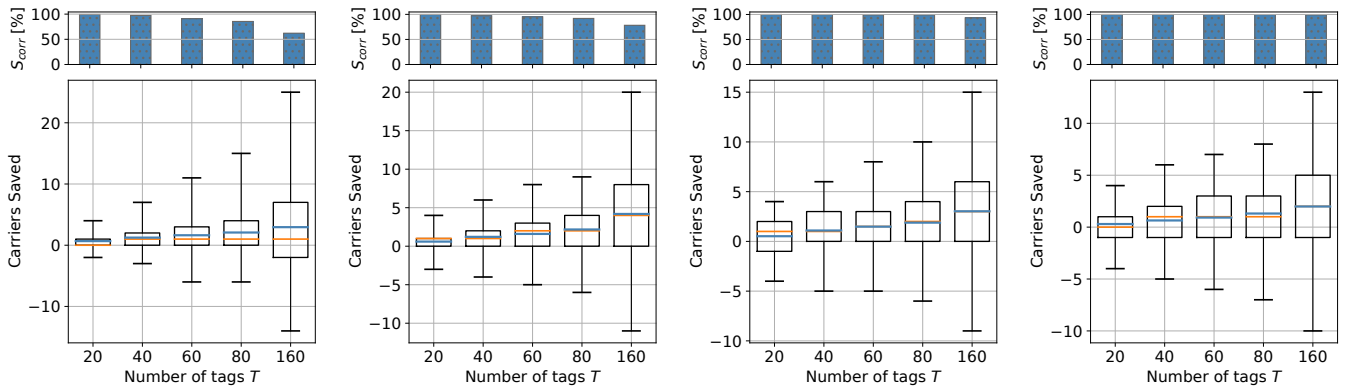
A determining factor for the real-world applicability of a scheduler is the computation time. Fig. 11 depicts a run time comparison of DeepGANTT and TagAlong on the same hardware. While TagAlong runs faster, the absolute values are so small that the difference is negligible in practice. Note that for the largest problem instances considered (60 nodes and 160 tags), the maximum runtime recorded was 1.49 s, with an average runtime of 0.25 s. This is in stark contrast with the optimal scheduler that takes several hours to compute schedules for just 10 nodes and 14 tags.

Time Complexity. In general, an attention-based message passing operation has time complexity $O(N+|E|)$, where N is the number of nodes in the graph and $|E|$ is the number of edges [49]. In the worst case, DeepGANTT performs T complete ML model passes, one for every tag in the wireless network. Hence, the complexity of our algorithm is polynomial in the input size: $O(T(N+|E|))$.

6.4 Performance on a Real IoT Network

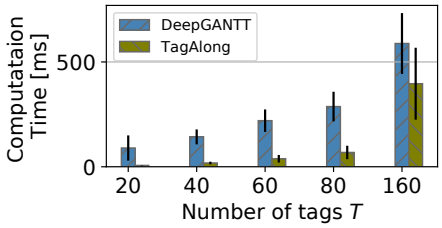
In this section we deploy DeepGANTT to compute tag interrogation schedules for a real IoT network and compare its performance with that of the TagAlong scheduler. We show that DeepGANTT can achieve up to 51.6% energy savings in tag interrogation.

Setup. We use an indoor IoT testbed consisting of 24 Zolertia Firefly devices (see Fig. 12a). The devices run the Contiki-NG operating system [38], communicate using IPv6 over IEEE 802.15.4 Time-Slotted Channel Hopping (TSCH) [7], and use RPL as routing protocol [54]. We collect the link connectivity among the IoT nodes

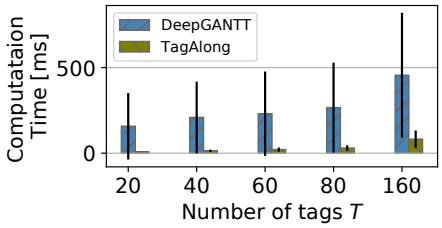


(a) 10-node topologies. **(b) 20-node topologies.** **(c) 40-node topologies.** **(d) 60-node topologies.**

Figure 10: *DeepGANTT outperforms TagAlong even when increasing the topology sizes far beyond those seen on training.* Scaling capabilities of DeepGANTT when compared to the TagAlong heuristic in terms of carrier savings. The model achieves a maximum carrier reduction of up to 43.42%, 43.86%, 33.93%, 32.14% for 160 tags and 10, 20, 40, and 60 nodes, respectively. The blue and the orange line represent the mean and the median, respectively. Box extents delimit 25 and 75 percentiles. Whiskers delimit 1 and 99 percentiles. The model also has a high success rate (S_{corr}), especially for larger topologies.



(a) Average runtimes for 10 nodes.



(b) Average runtimes for 60 nodes.

Figure 11: *While TagAlong runs faster than DeepGANTT, both run times are so small that the difference is negligible.* Run time comparison of DeepGANTT and TagAlong for 10 and 60 IoT nodes with various numbers of tags. The black bars represent the standard deviation.

every 30 min over a period of four days. We assume there is a link between any given pair of nodes if there is a signal strength of at least −75 dBm for carrier provisioning. The IoT network exhibits a dynamic change of node connectivity over the observed period of time (see Fig. 12b). We augment each of the collected network topologies with randomly assigned tags in the range $T \in \{10, 25, 50, 75, 85\}$, 100 assignments per T value.

Metrics. Apart from the *carriers saved* metric employed in Sec. 6.1 and 6.2, we also include the average energy per tag interrogation $\tilde{E}$, which is the total energy for tag interrogations E_{tot} divided by the number of tags in the network. Based on Fig. 4a, $\tilde{E}$ is given by:

$$\tilde{E} = \frac{E_{tot}}{T} = P_{tx}t_{tx} + P_{rx}\left(\frac{C}{T}t_{req} + t_{rx}\right) + P_{tx}\left(t_{req} + \frac{C}{T}t_{cg}\right), \quad (18)$$

where C is the number of carriers used in the schedule, T is the number of tags in the network, and both P_{rx} and P_{rx} correspond to the radio power at transmit and receive mode, respectively. We adopt $P_{rx} = 72mW$, $P_{tx} = 102mW$ based on the Firefly's reference values. Moreover, we assume $t_{req} = t_{tx} = 128\mu s$, $t_{rx} = 256\mu s$, and $t_{cg} = 15.75ms$ [41].

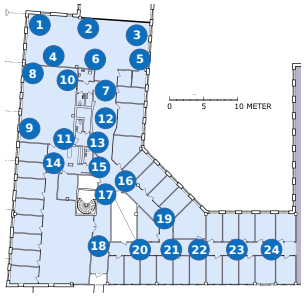
Results. Fig. 13 summarizes the results of deploying DeepGANTT to compute schedules on the real IoT network topologies compared to the TagAlong scheduler for different tag densities $\frac{T}{N}$. DeepGANTT shows a similar behaviour of *carriers saved* in Fig. 13a to those seen in Fig. 10. Moreover, our scheduler achieves average savings in $\tilde{E}$ compared to TagAlong above 10.96% for $\frac{T}{N} \geq 2.17$, with up to 51.64% maximum energy savings. Finally, Fig. 13b exhibits runtimes similar to those observed in Fig. 11.

7 DISCUSSION

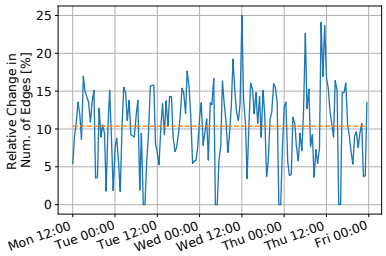
DeepGANTT is the first DL-based scheduler trained on optimal solutions which employs GNNs to generate interrogation schedules for tag-augmented IoT networks. Without the need for retraining, our scheduler can generate schedules for previously unseen and significantly larger topologies than those seen during training, while still achieving significant gains over the state-of-the-art heuristic. This section discusses practical implications and limitations of deploying DeepGANTT in real-world settings.

7.1 Practical Implications

Symmetry-Breaking. Our symmetry-breaking approach for generating the training data introduces bias in the scheduler: nodes with lower IDs would deplete their batteries faster, given that they will be selected more often as carrier generators. Far from a drawback, we believe this can be exploited as a feature at the application level. For instance, one could load-balance carrier scheduling

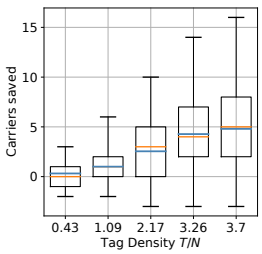


(a) *24 node IoT testbed layout.*

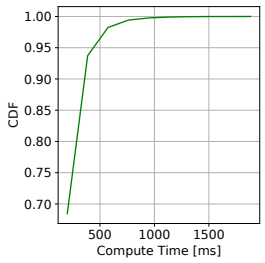


(b) *Link connectivity change over time.*

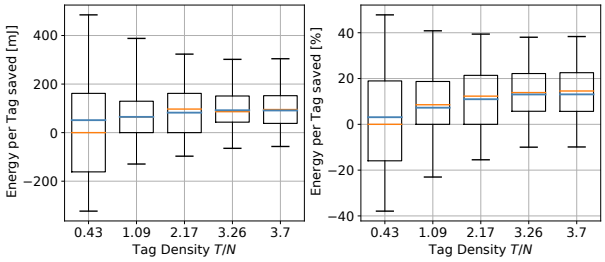
Figure 12: *We implement DeepGANTT to compute schedules for a real IoT network.* The network exhibits a high rate of change in its connectivity between two subsequent link collection time periods. The orange dashed line in 12b shows an average of 10.4%.



(a) *Carriers saved compared to TagAlong heuristic.*



(b) *DeepGANTT's runtime across all tag densities.*



(c) *Compared to TagAlong, DeepGANTT reduces per-tag energy consumption by* 13% *in average and up to* 51% *for high tag densities.*

Figure 13: *DeepGANTT achieves high energy savings compared to TagAlong, even for high tag densities.* We successfully deployed DeepGANTT to generate schedules for a real IoT network of $N = 24$ nodes and varying number of tags T. The blue and the orange line represent the mean and the median, respectively. Box extents delimit 25 and 75 percentiles. Whiskers delimit 5 and 99 percentiles.

over time by re-shuffling the node-IDs before each inference step, or schedule mains-powered carrier generators whenever possible instead of battery-powered ones simply by ordering the IDs in descending order of priority. Furthermore, the lexicographical minimization constraint is beneficial when the tags must be interrogated in a certain order, since our approach favors interrogating tags with lower IDs in the network early in the schedule.

Infrastructure Requirements. DeepGANTT is intended to run at an Edge/Cloud server to process requests from different IoT networks. While this induces a degree of centralization in the sense that the schedule is computed at one place, this is also true for the TagAlong heuristic [41], as both systems are equal in that regard. For DeepGANTT, this is motivated by the fact that it greatly benefits from hardware accelerators, such as a GPU. In such a case, it exhibits short runtimes as presented in Figures 11 and 13b.

Tracking Topology Changes. In our specific implementation, we leverage TSCH [7] and RPL [54] to gather the link state and node neighborhood information to build the topology graph and track changes thereof. In general, other alternatives for physical layer and routing protocols can be used for these purposes.

Schedule Dissemination. While there is certain communication overhead when disseminating the computed schedule for DeepGANTT, this is also true for the TagAlong heuristic, as both employ the same dissemination mechanisms. This overhead is proportional to the length of the schedule. As shown in Figure 8b, DeepGANTT produces in average shorter schedules than the heuristic, which implies a lower schedule dissemination overhead. Additionally, our system design does not restrict the IoT network from leveraging existing network flooding mechanisms, such as Trickle [29] or Glossy [10], as alternatives for disseminating the schedule.

7.2 Limitations and Future Work

Relative Tag-to-Host Location. Our system model assumes that tags are located in close proximity to their respective host, which allows us to leverage efficient carrier re-use as presented in Figure 3 and 4b. This is, is in general, a limitation of our system. However, this fits well with the envisioned scenario where the sensing capabilities of an unmodified network of COTS IoT devices are augmented with backscatter tags in a simple and scalable manner. While there are ways in which we may generalize the system to relax this assumption, this is beyond the scope of this work.

High-Mobility Scenarios. While we did not design DeepGANTT for highly mobile scenarios, the short runtimes of our scheduler allows it to operate in a timely manner and react fast to changes in the IoT network. Further work could evaluate deploying DeepGANTT for highly-mobile scenarios, and, if necessary, improve it to operate in such settings.

Tag Discovery Process. In our implementation, an IoT node does not automatically detect when a tag is added to, or removed from, its proximity. Rather, this must be explicitly notified to the IoT network. This is a general problem in the type of backscatter networks considered in this work, and out of the scope of this paper.

8 RELATED WORK

Our work is relevant both for scheduling in backscatter networks and for supervised ML applied to communications; in particular, to problems of a combinatorial nature. Many recent related efforts advance backscatter communications and battery-free networks [9, 12, 22, 24–26, 33, 37, 46, 60], but few of these address the efficient provision of unmodulated carriers. Pérez-Penichet et al. demonstrate TagAlong, a complete system with a polynomial-time heuristic to compute interrogation schedules for backscatter devices [41, 43]. Like our work, TagAlong exploits knowledge of the structural properties of the wireless network for fast scheduling. However, TagAlong's carefully designed algorithm produces wasteful suboptimal schedules. Van Huynh et al. [20] employ numerical analysis to optimize RF energy harvesting tags. By contrast, our work focuses on communication aspects and remains independent of the energy harvesting modality. Carrier scheduling resembles the Reader Collision Problem in RFID systems [17, 56, 58] in that both need to avoid carrier collisions. These works focus on the monostatic backscatter configuration (co-located carrier generator and receiver), whereas our work focuses on the bi-static configuration (separated carrier generators and receivers). The bi-static setting leads to a different optimization problem and our focus is on resource optimization rather than mere collision avoidance. Previous efforts in communications employ reinforcement learning with GNNs to solve combinatorial scheduling problems, mostly on fixed-size networks or static environments [2, 52, 59]. By contrast, our work focuses on a single solution tackling variable-size inputs and outputs, adequate for a multitude of varying conditions. Also novel in our work is that we employ supervised ML to solve the COP; to the best of our knowledge, this is a new approach within backscatter communications. Yet another novelty in our work is our strategy of restricting the solution space of the COP to boost the trained model's performance and scalability properties.

ML methods have been applied to COPs over graphs in the past years [50], for both reinforcement [5, 34] and supervised learning [30, 51]. Similar to our work, Vinyals et al. [51] implement an attention-based sequence-to-sequence model that learns from optimal solutions to solve the traveling salesperson problem. Likewise, Li et al. [30] employ GNNs [6, 28] to solve three traditional COPs with a supervised approach. Finally, we believe that our approach for dealing with multiple solutions in the scheduling problem could have far-reaching implications in solving the broad class of graph-related NP-hard COPs (such as traveling salesperson) using supervised ML techniques.

9 CONCLUSION

DeepGANTT generates interrogation schedules for a network of IoT devices interoperating with battery-free backscatter tags. Our scheduler leverages self-attention GNNs to overcome the challenges posed by the graph representing the problem and by the variable-sized inputs and outputs. Our symmetry-breaking strategy succeeds in training DeepGANTT to mimic the behavior of an optimal scheduler on small-sized network topologies. Without the need to be re-trained, our scheduler exhibits strong generalization capabilities to previously unseen problem instances up to six times larger than those used for training. DeepGANTT computes schedules that require on average 7%−10% and up to 50% fewer carriers than those produced by an existing, carefully crafted heuristic, even for the largest problem instances considered. More importantly, our scheduler performs within 3% of the optimal on the average number of carrier slots but with polynomial time complexity; lowering computation times from hours to fractions of a second. Our work advances the development of practical and more efficient backscatter networks. This, in turn, paves the way for wider employment in a large range of environments that today pose problems of great difficulty and importance.

ACKNOWLEDGMENTS

This work was financially supported by the Swedish Foundation for Strategic Research (SSF), by the European Union's Horizon 2020 AI@EDGE project (Grant 101015922), and by the Swedish Research Council (Grant 2017-045989). Nicolas Tsiftes was partially funded by KTH Digital Futures. The authors would also like to thank Erik Ylipää from RISE for early discussions about this work.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. In *Proc. Advances in Neural Information Processing Systems (NIPS) 2016 Deep Learn. Symp.* NIPS. arXiv:1607.06450

[2] Rajarshi Bhattacharyya, Archana Bura, Desik Rengarajan, Mason Rumuly, Srinivas Shakkottai, Dileep Kalathil, Ricky K. P. Mok, and Amogh Dhamdhere. 2019. QFlow: A Reinforcement Learning Approach to High QoE Video Streaming over Wireless Networks. *Proc Int. Symp. Mobile Ad Hoc Netw. Comput. (MobiHoc)*, 251–260. arXiv:1901.00959

[3] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Springer.

[4] Bluetooth SIG. 2021. *Bluetooth Core Specification 5.3*.

[5] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *Proc. Advances Neural Inf. Process. Syst. (NIPS)*, Vol. 2017-Decem. Neural information processing systems foundation, 6349–6359. arXiv:1704.01665

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proc. Advances in Neural Inf. Process. Syst. (NeurIPS)*. NeurIPS, 3844–3852. arXiv:1606.09375

[7] Simon Duquennoy, Atis Elsts, Beshr Al Nahas, and George Oikonomou. 2017. TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 11–18. https://doi.org/10.1109/DCOSS.2017.29

[8] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. *Benchmarking Graph Neural Networks*. Technical Report. arXiv:2003.00982

[9] Joshua Ensworth and Matthew S. Reynolds. 2015. Every smart phone is a backscatter reader: Modulated backscatter compatibility with Bluetooth 4.0 Low Energy (BLE) devices. In *Proc. Ann. Conf. RFID*. IEEE.

[10] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. 2011. Efficient network flooding and time synchronization with Glossy. In *Proc. 10th ACM/IEEE Int. Conf. Information Processing in Sensor Networks*. 73–84.

[11] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *Proc. ICLR Workshop Representation Learn. Graphs Manifolds*.

[12] Kai Geissdoerfer and Marco Zimmerling. 2021. Bootstrapping Battery-free Wireless Networks: Efficient Neighbor Discovery and Synchronization in the Face of Intermittency. In *(NSDI'21)*. 439–455.

[13] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. *Proc. 34th Int. Conf. Mach. Learn. (ICML)* 3 (apr 2017), 2053–2070. arXiv:1704.01212

[14] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[15] William L Hamilton. 2020. *Graph representation learning*. Vol. 14. Morgan & Claypool Publishers.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proc. Advances Neural Inf. Process. Syst. (NIPS)*, Vol. 2017-Decem. Neural information processing systems foundation, 1025–1035.

[17] Essia Hamouda, Nathalie Mitton, and David Simplot-Ryl. 2011. Reader Anti-collision in dense RFID networks with mobile tags. In *2011 IEEE International Conference on RFID-Technologies and Applications*. 327–334. https://doi.org/10.1109/RFID-TA.2011.6068657

[18] Mehrdad Hessar, Ali Najafi, and Shyamnath Gollakota. 2018. NetScatter: Enabling Large-Scale Backscatter Networks. In *NSDI'18*. USENIX.

[19] Josiah Hester and Jacob Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proc. 15th ACM Conf. on Embedded Netw. Sensor Syst.* (Delft, Netherlands) *(SenSys '17)*. Association for Computing Machinery, New York, NY, USA, Article 21, 6 pages. https://doi.org/10.1145/3131672.3131699

[20] Nguye Van Huynh, Dinh Thai Hoang, Dusit Niyato, Ping Wang, and Dong In Kim. 2018. Optimal Time Scheduling for Wireless-Powered Backscatter Communication Networks. *IEEE Wireless Commun. Lett.* 7 (2018), 820–823.

[21] IEEE. 2016. *IEEE Standard for Low-Rate Wireless Networks –Amendment 2: Ultra-Low Power Physical Layer.*

[22] Vikram Iyer et al. 2016. Inter-Technology Backscatter: Towards Internet Connectivity for Implanted Devices. ACM, 356–369. https://doi.org/10.1145/2934872.2934894

[23] Furqan Jameel, Ruifeng Duan, Zheng Chang, Aleksi Liljemark, Tapani Ristaniemi, and Riku Jantti. 2019. Applications of backscatter communications for healthcare networks. *IEEE Network* 33, 6 (2019), 50–57.

[24] Y. Karimi, A. Athalye, S. R. Das, P. M. Djurić, and M. Stanaćević. 2017. Design of a backscatter-based Tag-to-Tag system. In *2017 IEEE International Conference on RFID (IEEE RFID)*. 6–12. https://doi.org/10.1109/RFID.2017.7945579

[25] Bryce Kellogg et al. 2014. Wi-Fi Backscatter: Internet Connectivity for RF-powered Devices. In *Proc. Special Interest Group Data Commun. (SIGCOMM)*. ACM, New York, NY, USA, 607–618. https://doi.org/10.1145/2619239.2626319

[26] Bryce Kellogg et al. 2016. Passive Wi-Fi: Bringing Low Power to Wi-Fi Transmissions. In *Proc. Symp. Networked Syst. Des. Implementation (NSDI)*. NSDI, 151–164.

[27] Diederik P Kingma and Jimmy Lei Ba. 2015. Adam: A Method For Stochastic Optimization. In *Proc. Int. Conf. Learn. Representations (ICLR)*. arXiv:1412.6980v9

[28] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. 5th Int. Conf. Learn. Representations (ICLR)*. ICLR. arXiv:1609.02907

[29] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. 2011. The Trickle Algorithm. Retrieved Feb. 2023 from https://www.rfc-editor.org/rfc/rfc6206

[30] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *Proc. Advances in Neural Inf. Process. Syst. (NeurIPS)*. 539–548.

[31] Vincent Liu et al. 2013. Ambient Backscatter: Wireless Communication out of Thin Air. In *Proc. Special Interest Group Data Commun. (SIGCOMM)*. ACM, 39–50. https://doi.org/10.1145/2486001.2486015

[32] Jerry Ma and Denis Yarats. 2021. On the adequacy of untuned warmup for adaptive optimization. In *Proc. of the AAAI Conf. Artificial Intelligence*, Vol. 35. 8828–8836.

[33] A. Y. Majid, M. Jansen, G. O. Delgado, K. S. Yildirim, and P. Pawełłzak. 2019. Multi-hop Backscatter Tag-to-Tag Networks. In *Proc. Int. Conf. Comput. Commun. (INFOCOM)*. IEEE, 721–729. https://doi.org/10.1109/INFOCOM.2019.8737551

[34] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. 2020. Learning Heuristics over Large Graphs via Deep Reinforcement Learning. In *Proc. 34th Conf. Neural Inf. Process. Syst. (NIPS)*. arXiv:1903.03332

[35] Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT press.

[36] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. 2007. MiniZinc: Towards a standard CP modelling language. In *Lecture Notes in Computer Science*, Vol. 4741 LNCS. Springer Verlag, 529–543. https://doi.org/10.1007/978-3-540-74970-7_38

[37] P. V. Nikitin, S. Ramamurthy, R. Martinez, and K. V. S. Rao. 2012. Passive tag-to-tag communication. In *Proc. Int. Conf. RFID (RFID)*. IEEE, 177–184. https://doi.org/10.1109/RFID.2012.6193048

[38] George Oikonomou, Simon Duquennoy, Atis Elsts, Joakim Eriksson, Yasuyuki Tanaka, and Nicolas Tsiftes. 2022. The Contiki-NG open source operating system for next generation IoT devices. *SoftwareX* 18 (2022), 101089. https://doi.org/10.1016/j.softx.2022.101089

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, and et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proc. Advances Neural Inf. Process. Syst.*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.

[40] Carlos Pérez-Penichet, Frederik Hermans, Ambuj Varshney, and Thiemo Voigt. 2016. Augmenting IoT networks with backscatter-enabled passive sensor tags. In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MOBICOM)*. ACM, 23–27. https://doi.org/10.1145/2980115.2980132

[41] Carlos Pérez-Penichet, Dilushi Piumwardane, Christian Rohner, and Thiemo Voigt. 2020. A Fast Carrier Scheduling Algorithm for Battery-free Sensor Tags in Commodity Wireless Networks. In *Proc. Int. Conf. Comput. Commun. (INFOCOM)*. IEEE, 994–1003. https://doi.org/10.1109/infocom41043.2020.9155241

[42] Laurent Perron and Vincent Furnon. 2019. OR-Tools. https://developers.google.com/optimization/

[43] Carlos Pérez-Penichet, Dilushi Piumwardane, Christian Rohner, and Thiemo Voigt. 2020. TagAlong: Efficient Integration of Battery-Free Sensor Tags in Standard Wireless Networks. In *Proc. 19th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*. Sydney, Australia. https://doi.org/10.1109/IPSN48710.2020.00020

[44] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Trans. Neural Netw.* 20, 1 (jan 2009), 61–80. https://doi.org/10.1109/TNN.2008.2005605

[45] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. 2021. *Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification.* Technical Report. arXiv:2009.03509v5

[46] Vamsi Talla, Mehrdad Hessar, Bryce Kellogg, Ali Najafi, Joshua R. Smith, and Shyamnath Gollakota. 2017. LoRa Backscatter: Enabling The Vision of Ubiquitous Connectivity. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 3, 105:1–105:24. https://doi.org/10.1145/3130970

[47] Ambuj Varshney, Carlos Pérez-Penichet, Christian Rohner, and Thiemo Voigt. 2017. LoRea: A Backscatter Architecture That Achieves a Long Communication Range. In *Proc. 15th ACM Conf. Embedded Netw. Sensor Syst.* (Netherlands) *(SenSys '17)*. Association for Computing Machinery, New York, NY, USA, Article 50, 2 pages. https://doi.org/10.1145/3131672.3136996

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. Advances Neural Inf. Process. Syst. (NIPS)*, Vol. 2017-Decem. NIPS, 5999–6009.

[49] Petar Veličković, Arantxa Casanova, Pietro Liò, Guillem Cucurull, Adriana Romero, and Yoshua Bengio. 2018. Graph attention networks. In *Proc. 6th Int. Conf. Learn. Representations (ICLR)*. ICLR. arXiv:1710.10903

[50] Natalia Vesselinova, Rebecca Steinert, Daniel F Perez-Ramirez, and Magnus Boman. 2020. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access* 8 (2020), 120388–120416.

[51] Oriol Vinyals, Google Brain, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Proc. Advances Neural Inf. Process. Syst. (NIPS)*. 2692–2700.

[52] Fangxin Wang, Cong Zhang, Feng Wang, Jiangchuan Liu, Yifei Zhu, Haitian Pang, and Lifeng Sun. 2019. Intelligent Edge-Assisted Crowdcast with Deep Reinforcement Learning for Personalized QoE. In *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Vol. 2019-April. IEEE, 910–918. https://doi.org/10.1109/INFOCOM.2019.8737456

[53] Anran Wang et al. 2017. FM Backscatter: Enabling Connected Cities and Smart Fabrics.. In *NSDI'17*. USENIX, 243–258.

[54] T. Winter. 2012. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. Retrieved Oct. 2022 from https://www.rfc-editor.org/rfc/rfc6550

[55] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw.* 32, 1 (2021), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

[56] L. Yang, J. Han, Y. Qi, C. Wang, T. Gu, and Y. Liu. 2011. Season: Shelving interference and joint identification in large-scale RFID systems. In *Proc. Int. Conf. Comput. Commun. (INFOCOM)*. IEEE, 3092–3100. https://doi.org/10.1109/INFCOM.2011.5935154

[57] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware Graph Neural Networks. In *Proc. 36th Int. Conf. Mach. Learn. (ICML)*. arXiv:1906.04817v2

[58] H. Yue, C. Zhang, M. Pan, Y. Fang, and S. Chen. 2012. A time-efficient information collection protocol for large-scale RFID systems. In *Proc. Int. Conf. Comput. Commun. (INFOCOM)*. IEEE, 2158–2166. https://doi.org/10.1109/INFCOM.2012.6195599

[59] Han Zhang, Wenzhong Li, Shaohua Gao, Xiaoliang Wang, and Baoliu Ye. 2019. ReLeS: A Neural Adaptive Multipath Scheduler based on Deep Reinforcement Learning. In *Proc. Int. Conf. Comput. Commun. (INFOCOM)*, Vol. 2019-April. IEEE, 1648–1656. https://doi.org/10.1109/INFOCOM.2019.8737649

[60] Pengyu Zhang, Colleen Josephson, Dinesh Bharadia, and Sachin Katti. 2017. FreeRider: Backscatter Communication Using Commodity Radios *(CoNEXT '17)*. ACM, Incheon, Republic of Korea, 389–401. https://doi.org/10.1145/3143361.3143374