Ultra-Low Power DNN-based TSCH Scheduling at the Edge using the MAX78000

Martina Maria Balbi Antunes, Erman Okman, Elsa Lopez Perez, Blaz Korecic, Said Alvarado-Marin, Lance Doherty, Thomas Watteyne

Ultra-Low Power DNN-based TSCH Scheduling at the Edge using the MAX78000

Martina Balbi*,†, Erman Okman*, Blaz Korecic†, Elsa Lopez Perez†, Said Alvarado-Marin†, Lance Doherty*, Thomas Watteyne*,†

*Analog Devices
†Inria Paris
first.last@{analog.com*,inria.fr†}

***Abstract*—The optimization of Time-Synchronized Channel Hopping (TSCH) scheduling at the edge plays a critical role in enhancing the efficiency and reliability of the Internet of Things (IoT). In this paper, we propose an approach leveraging Deep Neural Networks (DNNs) for TSCH scheduling. Our approach aims to harness the capabilities of hardware accelerators such as the MAX78000 to achieve efficient and reliable scheduling. We train a DNN using a database generated with the Traffic Aware Scheduling Algorithm (TASA) and evaluate its performance in generating schedules, achieving an accuracy of over 94%. Our experiments demonstrate significant reductions in execution time, with the DNN-based approach achieving an average reduction of 99.69% compared to running TASA on the same microcontroller. Similarly, a substantial reduction in energy consumption is observed, with the DNN-based approach consuming only 5 μJ compared to 1,606 μJ by TASA running in software. This represents a significant milestone in the field low-power wireless networks, as it opens up new possibilities for implementing sophisticated AI-based solutions directly on edge devices, without the need for extensive computational resources.**

***Index Terms*—Deep Neural Networks, Embedded Artificial Intelligence, Low-Power Wireless Networks, Scheduling, TSCH**

I. Introduction

Efficient scheduling algorithms are essential for optimizing communication in the Internet of Things (IoT), where resource constraints and dynamic network conditions present significant challenges. Time-Synchronized Channel Hopping (TSCH) is a widely adopted protocol for enabling reliable and energy-efficient communication in such environments. Central to TSCH networks are scheduling algorithms that allocate transmission slots to network nodes, ensuring timely and efficient data delivery while minimizing energy consumption.

In recent years, there has been growing interest in leveraging Artificial Intelligence (AI) techniques to optimize TSCH scheduling. These algorithms offer the potential to learn complex patterns and relationships from data, enabling more adaptive and efficient scheduling decisions. However, deploying AI algorithms on resource-constrained devices presents unique challenges, requiring careful consideration of hardware limitations and computational efficiency. Notably, advancements in hardware accelerators, such as the MAX78000 microcontroller[1], have made it feasible to deploy AI algorithms on low-power devices. The MAX78000 is an advanced system-on-chip featuring an Arm® Cortex®-M4 and an ultra-low-power Convolutional Neural Network (CNN) accelerator.

In this paper, we present a novel approach to TSCH scheduling at the edge using Deep Neural Networks (DNNs), specifically tailored for deployment on the MAX78000. Our approach aims to train a DNN using a database generated with the well-known Traffic Aware Scheduling Algorithm (TASA) [1] and evaluate its performance in generating schedules with high accuracy and efficiency. Our main contribution lies in demonstrating the significant reduction in execution time and energy consumption achieved by running our DNN-based approach on an embedded device, compared to running the traditional software implementation of the algorithm.

Our work addresses specific gaps in the current state-of-the-art by demonstrating the feasibility and benefits of using DNNs for TSCH scheduling directly on ultra-low-power devices like the MAX78000. While previous studies have explored AI-driven approaches for wireless network scheduling, few have investigated the deployment of such algorithms on resource-constrained hardware. This approach not only harnesses the capabilities of the MAX78000 to significantly reduce execution time and energy consumption but also paves the way for the practical adoption of AI-based scheduling in real-world IoT applications. Our method showcases the transformative potential of hardware accelerators in enabling sophisticated AI algorithms to operate within the power and computational limits typical of edge devices.

Furthermore, the broader implications of this work extend beyond TSCH scheduling. The demonstrated efficiency and low energy consumption of our approach highlight the possibility of applying similar DNN-based techniques to other IoT protocols and resource-constrained environments. This could revolutionize the design and implementation of IoT networks, enabling more intelligent and adaptive systems that can function effectively at the edge, without relying heavily on centralized resources or cloud computing.

The remainder of this paper is organized as follows. Section II provides background information on TSCH scheduling algorithms. Section III details the proposed methodology of training a DNN with a TASA-generated database and adapting it to fit the limitations of the MAX78000. Section IV describes

[1] https://www.analog.com/en/products/max78000.html

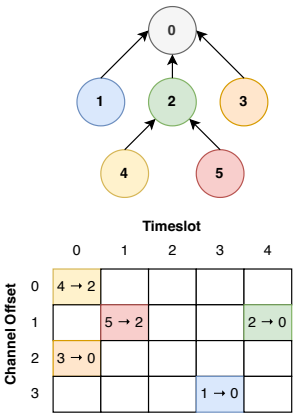

Fig. 1. A TSCH schedule (bottom) for a given topology (top).

the experimental setup used to evaluate the performance of the DNN-based scheduling approach on the MAX78000. Section V presents the results of the experiments, including analysis of execution time, energy consumption, and generated schedule validity. Finally, Section VI concludes the paper by summarizing the findings of this work and presenting insights obtained.

II. Background and Related Work

TSCH is a protocol in low-power wireless networks, introduced in IEEE 802.15.4e. In TSCH, time is divided into timeslots allocated for specific tasks. Nodes maintain synchronized timeslots, switching between frequency channels to reduce interference and improve reliability [2]. All nodes follow a shared communication schedule, determining actions in each timeslot: sending, receiving, or sleeping. An example of a possible schedule for a given network is depicted in Fig. 1. Network scheduling presents challenges as the TSCH specification lacks guidance on constructing and updating schedules. Numerous scheduling algorithms aim to optimize timeslot allocations and boost network performance.

TSCH scheduling algorithms are a crucial due to their role in orchestrating efficient communication among networked devices. Existing techniques for optimizing TSCH scheduling encompass a range of approaches tailored to enhance network efficiency and performance. Centralized algorithms, such as those by Palattela *et al.* [1] and Kin *et al.* [3], aim to achieve global optimization by considering the entire network topology and traffic patterns. These algorithms often rely on centralized controllers or coordinators to coordinate timeslot assignment and maintain synchronization among network nodes.

In contrast, distributed TSCH scheduling algorithms distribute scheduling decisions across network nodes, reducing reliance on central coordination and enhancing scalability. Distributed algorithms such as those proposed by Accettura *et al.* [4] and Morell *et al.* [5] often leverage local information, including neighboring node state and channel conditions, to autonomously adjust timeslot allocations.

Furthermore, autonomous techniques aim to decentralize scheduling decisions entirely, allowing individual network nodes to make independent scheduling choices based on local observations and objectives. These techniques grant nodes autonomy in adapting their transmission schedules without relying on centralized coordination or global optimization. Noteworthy examples of autonomous approaches include Orchestra [6] and the one proposed by Deac *et al.* [7].

AI techniques have been increasingly used for optimizing TSCH scheduling. These approaches leverage the capabilities of AI algorithms to learn from data and make intelligent decisions, aiming to improve network efficiency, reliability, and energy consumption. Notably, Reinforcement Learning (RL) has emerged as a popular choice among AI-based techniques for TSCH scheduling [8], allowing agents to learn optimal scheduling policies through interactions with the environment. Examples of algorithms leveraging RL to solve the scheduling problem for TSCH networks can be seen in works by Nguyen *et al.* [9] and Park *et al.* [10].

While RL has garnered significant attention for TSCH scheduling optimization, DNN approaches have been relatively less explored in the literature. Only a limited number of studies have attempted to use DNNs for wireless networks scheduling in general. Notably, a study presented by Morshedul *et al.* [11] stands out for its exploration of a DNN-based approach for TSCH networks. This study proposes a deep learning based scheme for TSCH network scheduling. It schedules the links of a slotframe using a maximum edge-weighted bipartite matching approach. The bipartite edge weight is designed to be composed of throughput and delay. The Hungarian algorithm is then employed to generate the training data, and a DNN is trained accordingly. Simulation results demonstrate that the proposed deep learning-based scheduling scheme achieves performance similar to the Hungarian algorithm-based scheduling scheme, with over 90% accuracy and nearly 80% execution time reduction.

Our DNN-based approach differs from RL-based methods in several key aspects. While RL is effective in scenarios where learning from interaction with the environment is feasible, it often requires significant computational resources and extensive training time to converge on an optimal policy, which can be challenging in resource-constrained environments like the edge. In contrast, our approach leverages a pre-trained DNN that directly maps traffic patterns to scheduling decisions, resulting in significantly faster inference times and lower computational overhead. This makes our method particularly well-suited for deployment on ultra-low-power devices and for applications where real-time scheduling decisions are critical.

For this work, we focus on TASA, introduced by Palattella *et al.* [1], as the scheduling algorithm that we aim to train a DNN with. TASA is a centralized scheduling algorithm that leverages established graph theory techniques to optimize network topology and traffic load. It is specifically designed as a traffic-aware method, meaning that it takes into account the network's traffic patterns and dynamics when making scheduling decisions. Within a centralized framework, TASA strategically employs matching and coloring procedures from graph theory to plan the distribution of timeslots and channel

offsets across the entire network topology graph. Operating typically on a gateway, TASA assigns timeslots and channel offsets across the entire network, aiming to minimize energy consumption and latency.

DNNs were chosen over RL and other AI techniques since they excel at capturing complex, non-linear relationships within data, making them ideal for modeling the complicated dependencies in TSCH scheduling. Additionally, the ability to train the DNN offline using a large dataset generated by the TASA algorithm allows for more controlled and predictable performance compared to RL, where the learning process might introduce variability. This ensures that the DNN can deliver high accuracy and low latency in generating schedules, which is essential for maintaining the reliability and energy efficiency of TSCH networks.

III. Proposed Method: Training a DNN with a TASA-generated Database

In this section, we present our proposed method for training a DNN with a database generated using a software implementation of TASA. Our objective is to investigate the feasibility of using DNNs to learn scheduling patterns from the TASA-generated database, and subsequently optimize TSCH scheduling at the edge.

A. Database Generation using TASA

We implemented TASA in Python and used it to generate 2 million TSCH network schedules, covering various traffic load scenarios. TASA was chosen for this study primarily because it is one of the most popular and well-established scheduling algorithms in the field. Its widespread use and established performance make it a reliable choice for generating a training database. However, it is important to note that the same procedure could have been applied using any other TSCH scheduling algorithm. The methodology we propose is not limited to TASA and can be adapted to work with different algorithms, depending on the specific requirements of the network or application. This flexibility allows our approach to be broadly applicable across various TSCH scheduling scenarios.

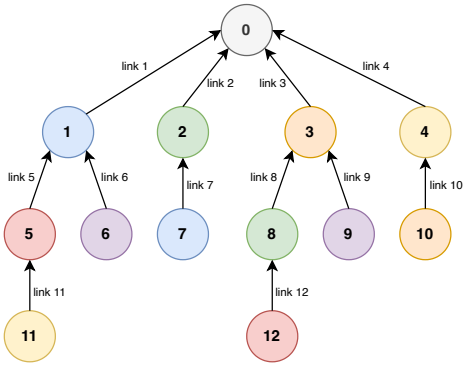

Fig. 2. Used network topology

To have a perfectly controlled environment, the samples are created for a single 13-node network with a fixed tree topology shown in Fig. 2. This fixed configuration ensures consistency across all generated schedules, with the only variable between samples being the traffic load of each node. We use this single topology to easily quantify the performance of our DNN approach; it does in no way mean this approach cannot be used on any other arbitrary topology.

However, it is important to acknowledge the potential limitations of using a single fixed topology in generating the dataset. While this approach simplifies the training process and enables controlled experimentation, it inherently restricts the generalization of the DNN to other network topologies. The DNN is trained to recognize patterns tailored to the selected topology, which may reduce its effectiveness in different network configurations. Despite these limitations, the primary focus of this work is to demonstrate improvements in time efficiency and energy consumption. Future work could address generalization challenges by incorporating multiple topologies into the training dataset, thereby enhancing the DNN's adaptability across diverse network scenarios.

The output of the TASA algorithm software implementation is a schedule represented as a matrix, where each row corresponds one of the 5 channel offsets used in the implementation, and each column represents one of the 40 timeslots in the slotframe. If a link is scheduled on a specific cell of this matrix, the algorithm sets that cell to the transmitting node's number. Otherwise the cell is assigned value 0. The used network's parameters are presented in Table I.

Given this setup, the TASA algorithm takes as input a vector of length 12 representing the total traffic load (denoted as q_i) for each node n_i in the network (excluding the sink n_0 which only receives packets) intended to be scheduled on a specific slotframe. The TASA algorithm processes this input vector and generates a corresponding schedule matrix that satisfies the network's communication requirements. A visual representation of the input and the output of the database generated by the TASA implementation is shown in Fig. 3. The numbers inside the input vector represent the traffic q_i of each of the nodes. If a specific link is scheduled at a certain timeslot and channel, then that cell is colored with the color of the link used in the input vector.

The primary objective of the TASA algorithm is to ensure that all traffic q_i from all nodes n_i reaches the designated sink node n_0 within the allocated slotframe duration. To achieve this, the algorithm schedules the links between nodes such that data transmission occurs efficiently, minimizing delays and ensuring timely delivery of packets.

Network Parameters	
Number of nodes	13
Number of links	12
Number of timeslots	40
Number of channels	5

TABLE I
Network Parameters

To create a diverse dataset for training the DNN, we generate 2 million random input vectors. Each input vector represents the traffic load of each node in the network, with q varying randomly from 1 to 9 packets. These random variations in traffic load simulate real-world scenarios where network traffic fluctuates dynamically.

These 2 million input vectors are passed to the Python implementation of TASA, which generates a 2 million sample database of input-output pairs. Each input-output pair consists of an input vector representing the traffic load of each node, and the corresponding output schedule generated by the TASA algorithm. This database forms the training dataset for the DNN, allowing it to learn the complex mapping between input traffic loads and optimal TSCH schedules.

B. DNN Architecture

The DNN architecture, tailored to fit within the constraints of the MAX78000 microcontroller, maintains a fixed input layer size of 12, representing the traffic load (q) of each node in the used TSCH network. Comprising an input layer, 3 hidden layers, and an output layer, the design emphasizes computational efficiency while retaining predictive accuracy. Each hidden layer incorporates the ReLU activation function to facilitate effective pattern learning within the network. The output layer produces a flattened schedule matrix of size 200, corresponding to the total number of timeslots multiplied by the number of channels.

The specific number of layers and neurons in our DNN architecture was carefully chosen to balance the trade-off between computational complexity and the need for high accuracy. Given the hardware constraints of the MAX78000, which include limited memory and processing power, we designed the DNN with only 3 hidden layers and a moderate number of neurons to ensure that the model remains lightweight enough to run efficiently on the microcontroller. Each hidden layer consists of 200 to 256 neurons, providing sufficient capacity for the network to learn complex patterns without exceeding the memory limitations. The choice of ReLU as the activation function was driven by its computational simplicity and effectiveness in deep learning tasks, which aligns with the need for fast inference times on low-power devices. This architecture was also selected based on preliminary experiments, where we evaluated various configurations to determine the optimal balance between model performance and hardware constraints.

C. Training Procedure

The DNN training is carried out using the PyTorch library[2], aligning with the MAX78000 pipeline[3]. The training process adopts a regression-like approach to minimize the Mean Squared Error (MSE) loss for optimal schedule prediction, facilitated by the ADAM optimizer [12].

The TASA-generated database comprising 2 million samples is pre-processed for training. These samples are divided into training and test sets, with an 80-20 split ratio. This data partitioning ensures a robust evaluation of the trained model's performance.

The DNN, implemented using PyTorch, undergoes training for 100 epochs. During each epoch, mini-batches of size 256 of training data are fed into the network. The network parameters are optimized iteratively using the ADAM optimizer. The objective is to minimize the MSE loss between the predicted and actual schedule matrices, enabling the network to learn and generalize from the training data effectively.

Hyperparameters such as learning rate, batch size, and the architecture of the DNN are fine-tuned to optimize training performance. A summary of the DNN architecture and hyperparameters is presented in Table II.

D. Adaptation to the MAX78000

Originally, the network was designed with more inner layers and a higher number of neurons in the hidden layers to enhance learning capabilities. However, the need for compatibility with the MAX78000 required a reduction in these parameters.

The limitations of the MAX78000, particularly regarding input/output data dimensions and weight memory capacity, has played an important role in shaping the adapted DNN architecture. For instance, input data had to be normalized from 0 to 1 for training with the MAX78000 training tool. Linear layers were specifically chosen to adhere to these constraints. Notably, the MAX78000 imposes a maximum dimension of 1023 for input or output data and supports up to 432 kB of kernel memory for weight storage. Adhering to these limitations, the adapted DNN architecture is crafted to optimize memory use and computational efficiency while ensuring compatibility with the MAX78000 hardware.

The adaptation process for deployment on the MAX78000 microcontroller involves several key steps to ensure seamless integration and optimal performance within the hardware constraints. Firstly, training is carried out using the MAX78000 *ai8x-training*[4] tools. The trained DNN model then undergoes quantization to reduce memory requirements and computational complexity. Quantization occurs after training, with the resulting decrease in accuracy evaluated on the test dataset to assess its impact.

The MAX78000 synthesizer tool, *ai8xize*[5], is designed to translate a trained PyTorch model into code executable on the

[2] https://pytorch.org/

[3] For more information on the MAX78000 pipeline refer to https://github.com/analogdevicesinc/ai8x-training.

[4] https://github.com/MaximIntegratedAI/ai8x-training

[5] https://github.com/MaximIntegratedAI/ai8x-synthesis

DNN Parameters	
Number of input features	12
Number of output features	200
Number of hidden layers	3
Number of input features in each hidden layer	200, 256, 256
Number of input features in output layer	256
Learning rate	0.01
Mini-batch size	256
Number of epochs	50

TABLE II
DNN PARAMETERS

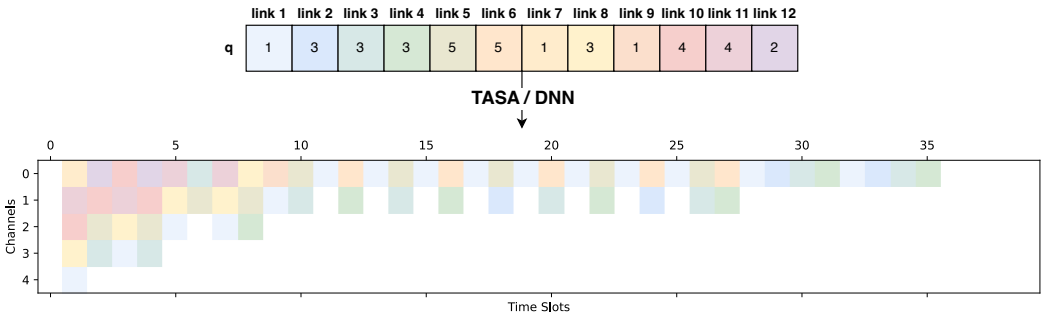

Fig. 3. Example of an input and output of the DNN or TASA

MAX78000 microcontroller. It requires a PyTorch checkpoint file and a YAML file describing the model's structure as inputs. Additionally, an example input data file is provided to verify the model's behavior on the hardware platform. The tool generates C code tailored for the MAX78000, which includes functions to load weights and input data, perform inference, and validate the classification results by comparing them with expected outputs.

E. Model Performance

This section provides an overview of the performance metrics obtained from the trained DNN models, including the test loss and accuracy for both the original model trained for a computer and the quantized model optimized for deployment on the MAX78000 microcontroller.

To measure accuracy, the test database is passed through the DNN models, and the output is rounded to the nearest integer and clamped between 0 and 12 to obtain valid links. The final schedule is then compared to the expected one by counting the number of cells that are correctly assigned, indicating the presence of the same link number. The accuracy is calculated as the sum of correctly assigned cells divided by the total amount of cells in the schedule. The MSE loss function is chosen to quantify the difference between predicted and actual outputs during model training.

The original DNN model trained for a computer achieves an accuracy of 95.98% with a MSE loss of 0.06981. These metrics indicate the high predictive accuracy and low discrepancy between predicted and actual outputs on the test dataset, highlighting the effectiveness of the model when generalizing.

The quantized DNN model optimized for deployment on the MAX78000 microcontroller achieves an accuracy of 94.87% with a MSE loss of 0.00646. Despite the quantization process and hardware constraints, the model still demonstrates strong predictive accuracy and minimal loss.

These results are summarized in Table III.

	Test Accuracy (%)	**Test Loss (MSE)**
DNN for computer	95.98	0.06981
Quantized DNN for MAX78000	94.87	0.00646[6]

TABLE III
DNN ACCURACY AND LOSS

IV. EXPERIMENTAL SETUP

This section outlines the experimental procedure conducted to compare the performance of the DNN-based approach with a software implementation of the TASA algorithm for TSCH scheduling optimization on the MAX78000 microcontroller. Through experimentation and comparative analysis of energy consumption and execution time, we gain insights into the efficiency and effectiveness of each method.

The experimental procedure involves deploying the trained DNN optimized for the MAX78000 microcontroller onto a MAX78000 Evaluation Kit[7]. Two sets of experiments are conducted to compare the performance of the DNN-based approach with the TASA algorithm implemented in C.

In the first set of experiments, the DNN is used to compute schedules for varying amounts of traffic load at the input nodes. The energy consumption and execution time of the DNN are measured for each computation. Additionally, a validity check is performed on the DNN's output to ensure it produces a valid schedule, meeting the requirements of the TSCH protocol.

In the second set of experiments, TASA is implemented in C and deployed on the MAX78000 microcontroller. Similar to the DNN experiment, schedules are computed for the same varying amounts of traffic loads at the nodes, and the energy consumption and execution time are recorded. These experiments serve as a benchmark for evaluating the performance of TASA when deployed on an embedded device, contrasting it against the DNN-based approach.

Following the experiments, a comparative analysis is conducted to evaluate the performance of the DNN-based approach against the TASA algorithm. The energy consumption and execution time of both methods are compared across varying traffic loads to assess their relative effectiveness in TSCH scheduling optimization. These are obtained using the embedded Power Monitor (PMON) available in the MAX78000

[6]Note that for training the quantized model, data had to be normalized to a value from 0 to 1, hence the difference in the loss' order of magnitude.

[7] https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/max78000evkit.html

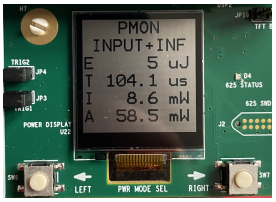

Fig. 4. Measurements taken of the DNN-based approach by the PMON

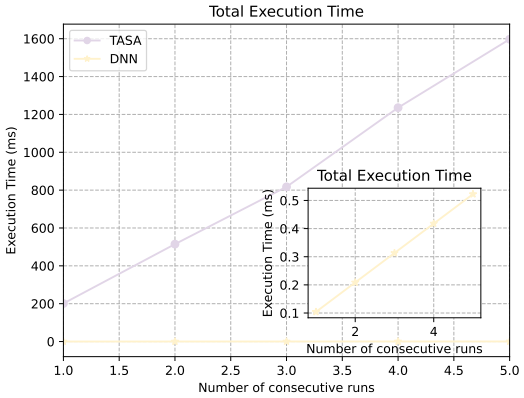

Fig. 5. Total execution time for both DNN version and TASA

Evaluation Kit. An example of a measurement taken with the PMON is shown in Fig. 4.

V. Results and Discussion

This section presents the results obtained from the conducted experiments and provides a discussion on the performance of the DNN-based approach compared to TASA for TSCH scheduling optimization at the edge using the MAX78000 microcontroller.

A. Execution Time

The total execution time required to calculate varying amounts of schedules using both TASA and the proposed DNN-based approach on the MAX78000 is presented in Fig. 5. Experimental results revealed that, for each technique, the execution time increases almost linearly with the number of schedules to be computed. However, a significant difference in execution time between the two techniques is observed, highlighting the efficiency gains offered by the DNN-based approach. In particular, the execution time for obtaining a single schedule using the DNN-based approach is measured at an astonishingly low 104 μs. In contrast, the C implementation of TASA requires substantially more time, 201.1 ms. This represents a 99.95% decrease in execution time when using the DNN-based approach.

B. Energy Consumption

Similar to the execution time comparison, the energy consumption of both TASA implemented in C and the proposed DNN-based approach is shown in Fig 6. Experimental results demonstrate a consistent trend, with the DNN-based

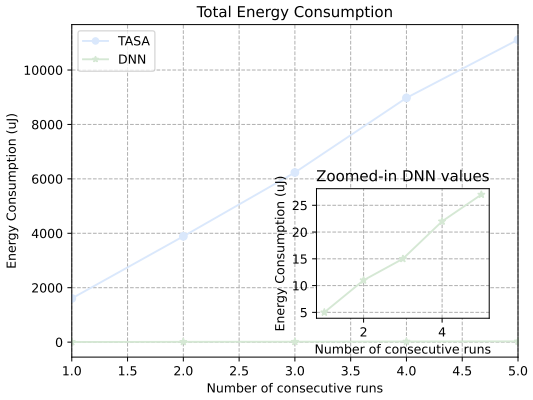

Fig. 6. Total energy consumption for both DNN version and TASA

approach consuming considerably less energy compared to the C implementation. Once again, the energy consumption for obtaining a single schedule using the DNN-based approach is measured at a mere 5 μJ. In contrast, TASA implemented in C requires 1606 μJ to compute just one schedule. This indicates a reduction of 99.69% in energy consumption when employing the DNN-based approach to obtain a schedule.

C. Validity of Generated Schedules

While the measured accuracy of the DNN-based model is high (94.87% for the quantized model running on the MAX78000), it is necessary to assess the validity of the generated schedules to ensure their reliability in real-world deployment scenarios. Even if a single link is wrongly scheduled, this can potentially lead to conflicts within the network. For instance, if link 1 (from n_1 to n_0, see Fig. 2) is scheduled instead of link 2 (from n_2 to n_0) in a particular timeslot, and concurrently, a link that interferes with link 1 (e.g. link 5 from n_5 to n_1) is correctly scheduled, a scheduling conflict arises. This is because n_1 cannot simultaneously transmit to n_0 and receive from n_1 in the same timeslot. While such wrong scheduling may seem trivial for the DNN, it renders the schedule invalid due to the resulting conflict, highlighting the importance of rigorous validation procedures. A portion of the test dataset is therefore passed through the DNN model, and the resulting schedules are scanned for conflicts. Out of the 10,000 schedules analyzed, only 6,147 presented no conflicts. This highlights the importance of further refining the DNN model and implementing advanced conflict resolution strategies to improve the validity and reliability of the generated schedules.

D. Discussion

The results of our experiments demonstrate the effectiveness of leveraging DNNs for TSCH scheduling optimization at the edge. With a measured accuracy of 94.87% for the quantized model running on the MAX78000, our DNN-based approach showcases promising capabilities in generating schedules with

high accuracy and efficiency. Moreover, the substantial reductions in both execution time and energy consumption highlight the transformative potential of DNN-based scheduling approaches in enhancing the performance of IoT at the edge.

Despite these promising results, several limitations and challenges need to be addressed. To begin with, while the accuracy of the DNN-based model is high, the validity of the generated schedules remains a critical issue. Our evaluation revealed that a significant portion of the generated schedules exhibited conflicts, which could compromise the reliability of the network. These conflicts typically arise when the DNN incorrectly schedules links in a way that leads to interference or simultaneous transmissions that violate TSCH protocol requirements.

To address this issue, potential strategies include integrating conflict detection mechanisms directly into the DNN's training process. One approach could be to augment the training data with information about potential conflicts and penalize the network during training when conflicts are detected. This would encourage the DNN to learn conflict-free scheduling patterns. Another approach could involve post-processing the generated schedules to identify and resolve conflicts before deployment.

Furthermore, the comparative results between the DNN-based approach and the traditional TASA algorithm show significant improvements in both execution time and energy consumption, with reductions of 99.95% in execution time and 99.69% in energy consumption. These results are particularly impressive in the context of edge applications, where power is a critical resource.

However, it is important to consider the trade-offs between accuracy and energy efficiency. While the DNN-based approach offers substantial energy savings during the inference phase on the MAX78000, this does not account for the energy consumption and time required for training the DNN. Training is typically conducted on more powerful systems that are not subject to the same resource constraints as the deployed edge devices. As such, the energy and time costs of training are usually less critical in the overall assessment, but they are still important to acknowledge. This distinction is crucial because the efficiency gains during deployment are the primary focus, especially for applications where the DNN model is trained once and used many times.

In edge applications, these trade-offs must be carefully balanced, as the benefits of lower energy consumption must be weighed against the potential risks of network unreliability due to invalid schedules. Future work could explore methods to fine-tune this balance, possibly by introducing adjustable parameters that allow for dynamic trade-offs between accuracy and energy consumption depending on the specific requirements of the application. For example, in scenarios where reliability is paramount, the system could be configured to prioritize accuracy, even at the expense of slightly higher energy consumption. Conversely, in situations where energy efficiency is the primary concern, the system might accept minor accuracy reductions in exchange for extended battery life or reduced operational costs.

Finally, scalability is another critical consideration for the deployment of DNN-based scheduling in diverse IoT networks, which can vary widely in size and complexity. While our current experiments utilized a fixed 13-node network topology to evaluate performance, real-world IoT deployments often involve much larger and more complex networks. As the network size increases, the input dimension to the DNN must scale accordingly, which can introduce challenges in terms of computational complexity and memory requirements, especially on resource-constrained devices like the MAX78000.

To address these scalability challenges, future research should explore the adaptation of the DNN architecture to support larger network sizes without sacrificing efficiency. Techniques such as model pruning, quantization, and the use of more sophisticated neural network architectures could be investigated to maintain a balance between accuracy and weight memory. Furthermore, developing adaptive DNN models capable of generalizing across various network topologies without requiring extensive retraining could significantly enhance the applicability of this approach to a wider range of IoT scenarios. These scalability improvements will be essential for ensuring that a DNN-based scheduling solution remains practical and effective as IoT networks continue to grow in complexity and scale.

VI. Conclusion

In this paper, we present a novel approach leveraging DNNs for TSCH scheduling at the edge, using the MAX78000 microcontroller. Our primary focus is on demonstrating substantial improvements in energy consumption and execution time through the integration of hardware accelerators, enabling efficient and reliable scheduling for IoT networks. Experimentation and analysis have provided valuable insights into the potential of DNN-based scheduling in resource-constrained environments.

The results highlight the strengths of our approach, with significant reductions of 99.95% in execution time and 99.69% in energy consumption compared to TASA. These outcomes showcase the feasibility of deploying sophisticated AI algorithms on ultra-low-power devices like the MAX78000 and how DNNs can deliver high efficiency during deployment, even on constrained platforms.

While the focus of this work is on efficiency, we acknowledge the limitation of our approach in terms of schedule validity. Some generated schedules exhibited conflicts that could impact network reliability. Addressing these conflicts was beyond the scope of this study and is identified as a key area for future research. Potential solutions include integrating conflict detection mechanisms into the DNN training process or employing post-processing techniques to eliminate conflicts in generated schedules.

Scalability also presents a challenge for real-world IoT deployments, where networks often exceed the size and complexity of our experimental setup. Future research should explore scalable DNN architectures and adaptive models capable of

generalizing to larger networks and diverse topologies without significant retraining. Additionally, extending the approach to other IoT protocols could demonstrate its versatility across a broader range of applications.

The broader impact of this work lies in its potential to influence the future design of IoT edge devices and networks. By demonstrating that DNNs can be effectively deployed on ultra-low-power devices like the MAX78000, this research paves the way for more intelligent and energy-efficient IoT systems. The methodologies and insights gained from this study could be adopted in other resource-constrained environments, leading to advancements in how AI is used at the edge.

REFERENCES

[1] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic Aware Scheduling Algorithm for Reliable Low-power Multi-hop IEEE 802.15. 4e Networks," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2012.

[2] T. Watteyne, A. Mehta, and K. Pister, "Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense," in *ACM Symposium on Performance Evaluation of Wireless Ad hoc, Sensor and Ubiquitous Networks (PE-WASUN)*, pp. 116–123, 2009.

[3] Y. Jin, P. Kulkarni, J. Wilcox, and M. Sooriyabandara, "A Centralized Scheduling Algorithm for IEEE 802.15. 4e TSCH Based Industrial Low Power Wireless Networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2016.

[4] N. Accettura, M. R. Palattella, G. Boggia, L. A. Grieco, and M. Dohler, "Decentralized Traffic Aware Scheduling for Multi-hop Low Power Lossy Networks in the Internet of Things," in *IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2013.

[5] A. Morell, X. Vilajosana, J. L. Vicario, and T. Watteyne, "Label Switching over IEEE802.15.4e Networks," *Transactions on Emerging Telecommunications Technologies*, vol. 24, no. 5, pp. 458–475, 2013.

[6] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2015.

[7] D. Deac, E. Teshome, R. Van Glabbeek, V. Dobrota, A. Braeken, and K. Steenhaut, "Traffic Aware Scheduler for Time-Slotted Channel-Hopping-Based IPv6 Wireless Sensor Networks," *Sensors*, vol. 22, no. 17, p. 6397, 2022.

[8] B. Cunha, A. Madureira, B. Fonseca, and J. Matos, "Intelligent Scheduling with Reinforcement Learning," *Applied Sciences*, vol. 11, no. 8, p. 3710, 2021.

[9] H. Nguyen-Duy, T. Ngo-Quynh, F. Kojima, T. Pham-Van, T. Nguyen-Duc, and S. Luongoudon, "RL-TSCH: A Reinforcement Learning Algorithm for Radio Scheduling in TSCH 802.15.4e," in *IEEE International Conference on Information and Communication Technology Convergence (ICTC)*, 2019.

[10] H. Park, H. Kim, S.-T. Kim, and P. Mah, "Multi-Agent Reinforcement-learning-based Time-Slotted Channel Hopping Medium Access Control Scheduling Scheme," *IEEE Access*, vol. 8, pp. 139727–139736, 2020.

[11] M. N. Morshedul Haque and I. Koo, "Throughput Optimization of IEEE 802.15.4e TSCH-Based Scheduling: A Deep Neural Network (DNN) Scheme," in *International Conference on Computer and Information Technology (ICCIT)*, pp. 844–849, 2022.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.