Efficient IoT Anomaly Detection via Graph-Based Neural Architecture Search

Nate Balmain
Center for Cyber Security and Privacy
University of Oregon
Eugene, OR, USA
balmain@uoregon.edu

Lei Jiao
Center for Cyber Security and Privacy
University of Oregon
Eugene, OR, USA
ljiao2@uoregon.edu

Jun Li
Center for Cyber Security and Privacy
University of Oregon
Eugene, OR, USA
lijun@uoregon.edu

***Abstract*—The rapid proliferation of Internet of Things (IoT) devices has introduced significant challenges in securing network infrastructure, stemming from device heterogeneity, dynamic communication patterns, and constrained computational resources. Traditional approaches to IoT network anomaly detection often rely on deep learning models that are too resource-intensive for edge deployment, while conventional Neural Architecture Search (NAS) techniques remain computationally prohibitive. To address these limitations, we propose a novel Graph-based Neural Architecture Search (GNAS) framework for efficient and adaptable model generation in IoT environments. By representing the search space as a graph, GNAS enables subgraph reuse and reduces redundant computation, significantly lowering training time compared to standard NAS methods. Models discovered through GNAS can be deployed at the edge, offering improved scalability and responsiveness. Experimental results on the NF-TON_IoT dataset demonstrate the effectiveness of our approach, with the best-performing subgraph model achieving 97% accuracy, 96% precision, 97% recall, and a 97% F1-score.**

I. Introduction

The growing presence of Internet of Things (IoT) devices in residential environments has introduced new levels of convenience and automation—ranging from smart light switches to programmable thermostats. However, this rapid proliferation, often outpacing adequate security measures, has made IoT ecosystems attractive targets for cyberattacks. Malware such as Mirai has been used to exploit vulnerabilities in these devices, incorporating them into botnets capable of launching DDoS attacks, conducting network reconnaissance, or facilitating lateral movement within home networks.

To address these threats, network-based anomaly detection has emerged as a promising approach for identifying compromised IoT devices by monitoring deviations in traffic patterns or behavior. While machine learning models have shown strong performance in this domain, their computational demands often exceed the capabilities of resource-constrained IoT hardware. Consequently, recent work has explored offloading detection to edge or cloud infrastructure, enabling the use of more complex models while maintaining scalability and real-time responsiveness.

This work was supported in part by the U.S. National Science Foundation under the grant CNS-2225949.

One challenge in IoT network anomaly detection lies in the heterogeneity of services and device behaviors. Anomalies in health monitoring systems may differ significantly from those in environmental sensing or industrial control, often requiring service-specific detection models. Even within a single service type, variations across device models, networks, and geographic regions hinder generalization. Maintaining separate models for each context quickly becomes impractical at scale, especially given the computational and memory constraints of edge devices. While Neural Architecture Search (NAS) has emerged as a promising solution for automating efficient model design, conventional NAS methods are typically resource-intensive—relying on exhaustive search and retraining—which makes them ill-suited for dynamic, resource-constrained IoT environments.

A second challenge stems from the sporadic and unpredictable communication patterns of IoT devices. Many operate intermittently or transmit data with low temporal density, necessitating anomaly detection models that can be dynamically activated and deactivated in response to device activity. However, this model management introduces overhead that can degrade system responsiveness and limit real-time threat detection. To address these challenges, we propose a framework for IoT anomaly detection that leverages Graph-based Neural Architecture Search (GNAS). By representing the architecture space as a graph, GNAS enables efficient exploration and reuse of shared components, significantly reducing training time while supporting adaptive, lightweight models suitable for dynamic edge deployments.

In the remainder of this position paper, we:

- Present background information and review related work in the domains of IoT network anomaly detection and neural architecture search (section 2).
- Describe our proposed methodology, including the design of a graph-based search space, the training procedure, and the discovery and selection of high-performing subgraph architectures (section 3).
- Evaluate the performance of our approach through preliminary experiments, demonstrating improved efficiency over conventional disk-based model switching. Our framework maintains a low memory overhead, with the full supernet requiring less than 10 MB in the worst

case. The best-performing discovered subgraph achieves 96% precision, along with 97% accuracy, recall, and F1-score (section 4).
- Conclude with a discussion of our findings and their implications for future work (sections 5-6).

II. Background and Related Work

A variety of techniques have been proposed for anomaly detection in IoT networks, ranging from classical machine learning to deep learning methods. Deep neural networks (DNNs) have shown strong performance in modeling the complex, high-dimensional patterns in network traffic, while recurrent architectures—particularly long short-term memory (LSTM)—further improve detection by capturing temporal dependencies [1]–[4].

Neural Architecture Search has emerged as a method to automate model design, but traditional approaches often rely on brute-force strategies that instantiate, train, and discard a large number of candidate architectures. This leads to prohibitive training times and poor scalability, especially in resource-constrained settings like IoT. The search space typically spans multiple dimensions—activation functions, layer types, kernel sizes, and connectivity patterns—causing a combinatorial explosion in candidate models as parameter complexity grows. These inefficiencies make naïve NAS unsuitable for environments that require lightweight, rapidly deployable models.

A. Efficient Network Architecture Search

Recent advances in NAS have increasingly focused on improving efficiency by reducing the need to train a large number of individual models from scratch. Traditional NAS approaches—such as those relying on reinforcement learning (RL) controllers or hypernetworks—often incur significant computational costs that are prohibitive for real-time or resource-constrained deployments. However, Bender et al. [5] showed that neither an RL controller nor a hypernetwork is strictly necessary to achieve strong performance, which aligns well with our design: we avoid the use of auxiliary controllers, enabling a more lightweight and interpretable search strategy suitable for cloud-edge deployment scenarios.

A common insight across these works is that training and discarding full models is inherently inefficient. Any strategy that enables parameter reuse or shared optimization across candidate architectures can lead to a more scalable and cost-effective search process. Notably, Efficient Neural Architecture Search (ENAS) [6] introduced a weight-sharing supernet framework in which subgraphs are sampled and trained within a shared network, significantly reducing search time and computational cost. Our approach extends this principle to graph-structured supernets in which subgraphs overlap at varying depths, promoting shared optimization across a broad architectural space.

Similarly, the Once-for-All (OFA) framework [7] introduces a progressive shrinking technique and component reuse strategy, enabling architectural flexibility with minimal retraining. While OFA explicitly incorporates elastic depth—retaining only the first L layers of deeper networks and bypassing the rest via residual connections—our framework benefits from a similar effect implicitly. This emerges as a byproduct of our design, which encourages overlapping sub-networks of varying depths. As a result, we achieve implicit weight sharing and parameter efficiency, with natural skip connections emerging across the search space.

Our method is particularly suited for cloud-edge inference scenarios, where memory and latency constraints still exist, but training and architecture search can be conducted offline with a higher resource budget. This allows us to balance search efficiency with the need for flexible, task-specific model specialization.

B. Graph Network Architecture Search

The Gradient-based Differentiable Architecture Sampling (GDAS) method [8] models the neural architecture search space as a directed acyclic graph (DAG), enabling the discovery of both normal and reduction cells that can be stacked to construct full networks. By leveraging differentiable sampling techniques, GDAS efficiently explores a wide range of subgraph architectures, significantly reducing search time—requiring only a few GPU hours on datasets such as CIFAR-10.

While our approach similarly represents the search space as a graph, it differs in both scope and objective. Instead of focusing on convolutional networks, we target the discovery of hybrid LSTM-DNN architectures. Additionally, rather than optimizing a single architecture, our framework is designed to train multiple overlapping sub-networks concurrently, thereby constructing a parameter-efficient, multi-model supernet.

DySR: Adaptive Super-Resolution via Algorithm and System Co-Design (DySR) [9] served as one source of inspiration for this work. The authors demonstrated efficient exploration of neural architecture search spaces by leveraging techniques such as weight sharing and graph-based search space representations, significantly reducing training time for image super-resolution tasks. In addition, they introduced "network modules"—an individual section or block of a machine learning model. Our work adopts a similar concept of modular design, incorporating both DNN and LSTM modules. These modules are instantiated concurrently and explored efficiently within the supernet framework.

Differentiable Architecture Search (DARTS) [10] models each cell as a DAG with edges representing candidate operations. It relaxes the discrete search space into a continuous one by assigning learnable mixture weights to all operations, enabling gradient-based optimization of connectivity patterns. In contrast, our method samples and trains complete architectures with shared parameters, preserving architectural diversity without continuous relaxation.

III. Methodology

To support efficient exploration of neural architectures, we implement a systematic subgraph generation strategy for constructing modular neural network components tailored to different configurations. The method dynamically constructs sub-networks composed of shared DNN and LSTM blocks based on specified hyperparameters:

- N: The number of hidden DNN layers
- I: The Input size of each DNN layer
- O: The Output size of each DNN layer
- L: The number of LSTM Layers

All modules use ReLU as the activation function. Additionally, a dropout rate of 0.2 is applied to both the output of the model and the LSTM-to-DNN interface to encourage regularization and prevent overfitting.

The resulting modules are assembled sequentially to form a distinct subnetwork, which is then stored using a unique identifier derived from its parameter configuration. This design allows for modular instantiation, where each subgraph can be independently reconstructed or selected based on its parameter set. The modular architecture promotes scalable architecture search by enabling systematic variation of key structural parameters while reusing shared components across multiple subgraphs.

Figure 1 illustrates the supernet construction process using a simplified configuration, where the hyperparameters are defined as $N \in \{1, 2\}$ $I = O$, and $L \in \{1, 2\}$. For clarity, the fourth generated subgraph (comprising 2 LSTM modules and 1 DNN module) is omitted from the figure. Because $I = O$, no structural variations in the DNN layer dimensions are introduced in this example.

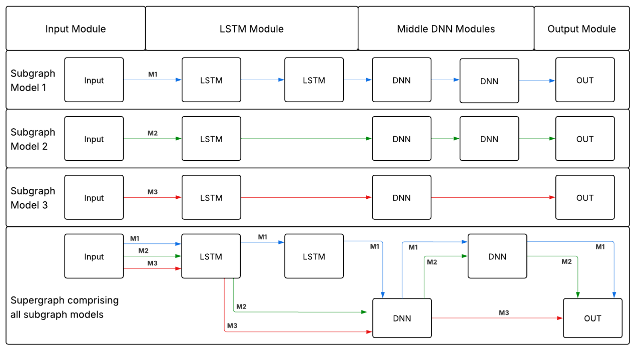

Fig. 1. The supergraph (bottom) is the combination of the subgraph models M1, M2, and M3 (above). Colored lines show the path of data through the model

Each subgraph corresponds to a distinct architecture instantiated by the Generator from the specified hyperparameters. When a subgraph is selected for training or inference, data is routed exclusively through its associated layers. For instance, if subgraph M3 is selected, the input passes through its LSTM module and the associated DNN module (a single layer in this case), and is then forwarded to the classification output.

Importantly, this architecture naturally incorporates skip connections, enabling efficient training. Nodes and connections not utilized by the active subgraph are bypassed, and their weights remain unchanged during backpropagation. Conversely, any node involved in the forward pass contributes to gradient updates and, by extension, to the partial training of all subgraphs that share those components.

During training, a single subgraph is sampled uniformly at random from the set of candidate architectures defined within the supernet. The selected subgraph is instantiated and trained on a batch of data using standard backpropagation. Crucially, only the parameters associated with the active subgraph are updated during each iteration, enabling efficient weight sharing across the larger architecture space. Conceptually, this can be viewed as "switching on" the selected subgraph in memory while "switching off" the others.

This parameterization serves as a means to constrain the size of the supernet, reducing the number of models to train and, consequently, the computational cost during the training phase. While this approach shares similarities with traditional Neural Architecture Search, it introduces a trade-off: fewer models are trained, resulting in a smaller pool of candidate models for deployment. The primary objective of this work is not to exhaustively explore every possible model configuration or to evaluate all their performances—an infeasible task—but rather to demonstrate the feasibility of this approach for network edge applications.

One of the known limitations of weight-sharing supernet training is that poorly performing subgraphs may override shared parameters that are critical to better-performing models, potentially degrading overall performance [5] [11]. While this is a valid concern, in our experiments we found that it did not significantly impact final model quality—likely due to the relatively low number of searched models and the regularizing nature of parameter reuse across the subgraph space.

A more prominent issue arises from the imbalance in training frequency across layers: early layers, which are shared across many subgraphs, receive substantially more updates than later layers that are only present in deeper or more complex architectures. As a result, smaller models with greater layer overlap tend to converge more quickly, while larger or sparsely overlapping models train more slowly and may underperform without intervention.

To further mitigate this imbalance, the following strategies can be employed: 1: Oversampling larger models during training to increase the update frequency of deeper layers, and 2: Biased subgraph training toward higher-performing subgraphs, ensuring that well-performing components receive proportionally more updates. These techniques help balance the learning dynamics across the supernet and reduce convergence disparities between simple and complex architectures.

A. *Model Selection*

Loading the full supernet into memory can be prohibitively resource-intensive, even on capable edge servers. This is due to the cumulative size of the network and the presence of many overlapping subgraphs that offer marginal improvements in detection performance or computational efficiency. To mitigate this, we adopt a sample-and-bin strategy, inspired by DySR [9], aimed at reducing both the memory and runtime overhead of the final deployed supernet.

In this approach, each distinct subgraph is evaluated and categorized into bins based on three criteria: (1) overall classification accuracy, (2) its F1 score on specific attack types, and (3) memory footprint, measured by model size. This evaluation yields three primary subgraph categories: general-purpose models with strong overall accuracy, specialized models optimized for detecting specific threats, and high-capacity models that offer superior performance but may exceed edge deployment constraints.

Subgraph selection is then guided by deployment-specific priorities. For instance, a high-accuracy generalist model can serve as a gatekeeper to direct inference requests, or be deployed standalone if its performance meets operational thresholds. Depending on available hardware or the need to prioritize certain threat classes, the final supernet can be constructed by selecting top-performing subgraphs from each bin. Because each subgraph is defined within a parameterized search space, selected architectures can be easily re-instantiated or fine-tuned as needed.

In our implementation, we construct a reduced supernet by selecting the best subgraph for each attack class based on F1-score, as well as the top 10 subgraphs ranked by overall accuracy. The architectural parameters (N, I, O, L) of these subgraphs are passed to the Generator, which instantiates only these specific configurations—excluding all other candidates from the original supernet.

The degree of parameter sharing in the reduced supernet depends on structural overlap among selected subgraphs. Greater overlap leads to more efficient weight sharing, while more disjoint architectures reduce this benefit. The reduced supernet is trained from scratch, with each subgraph receiving dedicated training until convergence.

This selective strategy results in a compact, efficient supernet that maintains strong detection performance while reducing computational overhead. The final model is well-suited for deployment in resource-constrained edge environments, offering a practical balance between performance and efficiency.

B. Model Switching and Deployment

Retaining the full supernet in memory enables the system to leverage the specialized strengths of individual subgraphs, each tailored to detect specific types of network anomalies. However, realizing this benefit requires an effective mechanism to dynamically activate the most appropriate subgraph based on input context. We propose two such model-switching strategies: one that relies on a traditional Network Intrusion Detection System (NIDS), and another that uses a lightweight, learned model-selection policy.

In the NIDS-based switching strategy, an external system such as Sunblock [12] monitors traffic between IoT devices and the central server. While capable of independent threat detection, the NIDS also serves as a gatekeeper that forwards ambiguous or uncertain traffic to the supernet. Based on traffic characteristics—such as packet frequency, payload size, or protocol distribution—it can route inputs to the subgraph most suited for the suspected threat class (e.g., DDoS, MitM, or injection attacks). This hybrid model combines the speed and precision of signature-based detection with the adaptability of machine learning-based anomaly detection. Optionally, the supernet—or a subset of subgraphs—may be embedded within the NIDS as an auxiliary module.

Alternatively, a model-based switching mechanism deploys a lightweight controller alongside the supernet. This controller is not optimized for accuracy but rather for efficiency and class discrimination. When its confidence exceeds a predefined threshold, it triggers the subgraph most appropriate for the predicted anomaly class. This approach allows the system to specialize inference dynamically, executing only the relevant subgraph, thus minimizing overhead while maintaining responsiveness.

An additional variant integrates the controller directly into the supernet. A general-purpose anomaly detector first screens incoming data, and if an anomaly is detected, the input is routed through the most appropriate subgraph using a simple rule—e.g., if the probability of class X exceeds threshold T, activate subgraph Y. This enables selective specialization while maintaining overall detection robustness, making it particularly effective in time-sensitive or resource-constrained environments.

Deployment of the compact supernet is intended at the cloud edge, which provides a vantage point for observing traffic between IoT devices and central servers. This placement supports real-time detection while minimizing latency. However, edge devices typically operate under tight resource constraints. We target a baseline deployment environment with modest specifications—2 GB RAM, dual-core CPU, and 128 GB storage—comparable to a low-end laptop. The framework remains scalable: in settings with more powerful hardware, multiple subgraphs can be run in parallel to further enhance detection throughput and coverage.

IV. Evaluation

All model training and evaluation were conducted on Google Colab [13], utilizing the following hardware configuration: a CPU with 4 cores and 8 threads, 51 GB of RAM, and an Nvidia Tesla T4 GPU with 15 GB of VRAM. During testing, the GPU usage never exceeded 1 GB, likely due to the relatively small search space. A single epoch of primary supernet training took approximately 3 minutes and 30 seconds, whereas training the deployment supernet took approximately 1 minute 30 seconds–primarily due to the lower number of subgraphs which need to be trained for deployment. Although this hardware exceeds the target specifications, the compact supernet remains deployable on the intended platform.

Testing and inferrence was performed on the NF-TON_IoT dataset [14], an enhanced version of the widely used TON_IoT benchmark [15], designed for IoT network intrusion detection. Data preprocessing involved scaling numerical features using standard scaling and organizing flow-level records into time-series sequences of length 50. The records were sorted by the flow start timestamp and grouped by the IPv4 source address. Each flow was defined by a 5-tuple: source and destination IP addresses, source and destination ports, and protocol (TCP/UDP). The predominant attack type (mode), if present, was used as the training label, with no attack assigned in the absence of an attack.

For model optimization, the ADAM optimizer was employed with a learning rate of 0.0001, using cross-entropy loss as the objective function. This configuration was selected for its proven effectiveness in classification tasks involving deep learning. Additionally, to address the issue of class imbalance—particularly the dominance of DDoS-related flows—the number of flow samples per class was capped to 200,000. This balancing strategy was employed to mitigate model bias toward the majority class and to ensure that the classifier maintained sensitivity to less frequent but equally critical types of anomalous behavior.

In order to evaluate model performance, the standard ML metrics of Accuracy, Precision, F1-Score, and Recall results are shown in Table 1 where all subgraph models were trained for 15 epochs each, for a total supernet training epoch length of 210 (15 epochs * 14 subgraphs). In order to achieve a fair comparison, two state of the art (SOTA) models, SOTA1 from [1] and SOTA2 from [2], were trained for 210 epochs each. These models are used to assess how deep neural networks without LSTM components perform under limited training budgets and when applied to time series data. Further, we included traditional machine learning classifiers such as Random Forest (RF), Support Vector Machine (SVM), Naive Bayes (NB), and Logistic Regression (LR). Our subgraph models are characterized by their structure in the format $N\_I\_O\_L$, where: N = Number of DNN layers, I = DNN input size, O = DNN output size, L = Number of LSTM layers respectively.

A. Results

Table 1 compares the evaluation of several classification models and individual supernet subgraph models. The best-performing configuration, 1_128_128_1, which consists of a single DNN layer with input and output size 128 and one LSTM layer, achieved the highest accuracy (0.9723) and F1-score (0.9701), demonstrating strong temporal and structural modeling with a relatively simple architecture.

Model	1_128_128_1	RF	1_32_32_1	SVM	NB
Accuracy	0.9723	0.96	0.7518	0.7	0.6
Precision	0.9684	0.96	0.7489	0.64	0.55
Recall	0.9723	0.95	0.7518	0.6	0.51
F1-Score	0.9701	0.95	0.7494	0.6	0.57
Model	1_32_128_5	LR	20_64_128_5	SOTA2	SOTA1
Accuracy	0.2405	0.12	0.2443	0.1235	0.1235
Precision	0.1764	0.1	0.0597	0.0152	0.0152
Recall	0.2405	0.02	0.2443	0.1235	0.1235
F1-Score	0.1116	0.09	0.096	0.0271	0.0271

TABLE I
Performance metrics for SuperNet subgraphs, traditional ML models, and SOTA baselines after 210 training epochs. Subgraphs with overlapping DNN input/output dimensions perform well, while those with sparse overlap or higher complexity underperform under limited training.

Among traditional models, Random Forest performed competitively (F1 = 0.95), while SVM and the simpler DNN-LSTM subgraph 1_32_32_1 showed moderate effectiveness (F1 $\approx$ 0.75). In contrast, deeper or wider architectures such as 1_32_256_1 and 20_64_128_5 suffered from reduced performance (F1 $<$ 0.25), likely due to overfitting and limited parameter updates under shared training.

State-of-the-art baselines (SOTA1 and SOTA2) underperformed significantly (F1 $\approx$ 0.027), highlighting the importance of temporal modeling and architectural simplicity in constrained training scenarios typical of edge-oriented IoT environments. Overall, our results suggest that lightweight DNN-LSTM subgraphs with balanced design offer a strong trade-off between performance and deployability for IoT anomaly detection.

Memory efficiency is a critical factor in this work, as any performance advantage from using multiple subgraphs could be undermined by excessive model size at deployment. Figure 2 illustrates the relative memory cost of loading the supernet under four configurations: using the smallest, final selected subgraphs, average, and largest subgraphs. Notably, even in the most memory-intensive case—where the supernet includes the largest subgraphs—the total memory footprint remains modest at approximately 10 MB. By contrast, a reduced subnet comprising 10 average-sized subgraphs incurs a memory cost of just over 5 MB. Although the final deployed model is 2.45MB large, this is only slightly larger than the memory footprint of the two state-of-the-art models combined (1.49MB). However, the final deployed supernet offers significantly greater flexibility, allowing adaptive inference and selective specialization without requiring full model reloading.

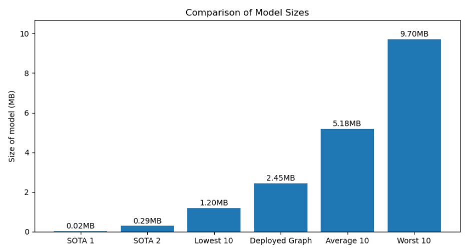

Fig. 2. Bar chart illustrating the relationship between in-memory size and subgraph characteristics, including the two state of the art DNN approaches SOTA1 [1] SOTA2 [2], the smallest in-memory subgraphs, the size of the final deployed metagraph, the average size of 10 subgraphs, and the 10 largest in-memory subgraphs.

Model-switching latency is another critical performance consideration, particularly for real-time or resource-constrained deployments. Figure 3 presents the average time (in milliseconds) required to switch between models under three scenarios: (1) full model swapping from disk, (2) full model switching from memory, and (3) subgraph switching within the supernet. Each measurement was repeated 10,000 times to provide statistically robust results. The bars represent mean switching times, and error bars indicate standard deviation. As expected, loading models from disk incurs the highest latency and presents a significant bottleneck to responsiveness. In contrast, switching between preloaded models in memory incurs minimal overhead and has a relatively small impact on system performance. Swapping between subgraphs within the supernet achieves a favorable trade-off, with switching times that are lower than disk-based loading but slightly higher than in-memory full model swaps. However, this approach also exhibits a higher standard deviation in latency (0.0830 ± 0.0542 ms), reflecting variability introduced by subgraph complexity and shared component reuse. Overall, these results highlight the efficiency and adaptability of the supernet approach: while slightly more variable, it avoids the substantial cost of disk-based swapping and enables dynamic specialization at runtime with minimal delay.

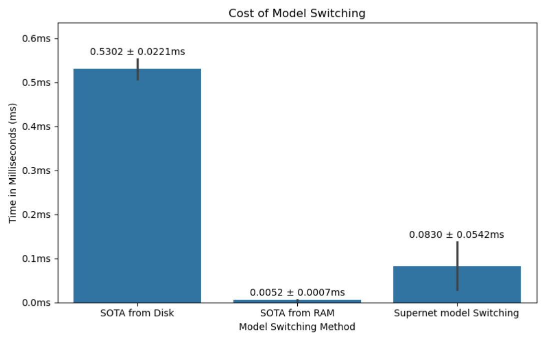

Fig. 3. The cost of swapping distinct models from Disk, distinct models from RAM, and swapping subgraph models

V. Discussion and Future Work

One important observation from our experiments is that the training process disproportionately favors subgraphs with high structural overlap, particularly in the early layers. These frequently reused components receive more updates, leading to faster convergence and better overall performance for the most commonly selected subgraphs. Interestingly, we found that the issue of inferior models overwriting shared weights had minimal adverse impact on the final performance of well-performing subgraphs. This suggests that parameter sharing, even across suboptimal architectures, can be both efficient and robust when managed within a constrained design space.

A major strength of our approach lies in its efficiency during training: within a limited number of epochs, our method is able to identify several high-performing candidate architectures. Compared to training each candidate model individually, the supernet not only discovers the best-fitting architectures, but also trains them more effectively due to the cumulative benefit of weight sharing across overlapping subgraphs.

Future work will focus on adaptive sampling strategies to better balance training across underrepresented or structurally unique subgraphs. Additionally, further exploration into dynamic weight partitioning, controller-based switching, and activation sparsity could enhance model specialization without compromising efficiency. Extending the current framework to support multi-objective search, such as balancing accuracy, memory usage, and inference time, is also a promising direction for deployment on resource-constrained IoT devices.

VI. Conclusion

In this work, we addressed the challenges of anomaly detection in IoT networks by proposing a novel approach based on Graph-based Neural Architecture Search (GNAS). IoT environments present unique difficulties due to the heterogeneity of device behaviors, dynamic communication patterns, and resource constraints at the network edge. Traditional NAS methods, while powerful, are often too computationally intensive to be practical in these settings. Our GNAS framework offers a more efficient solution by modeling the architecture search space as a graph, reducing redundant computation and enabling the reuse of overlapping subgraphs.

Our GNAS approach mitigates this limitation by representing the architecture search space as a graph, enabling efficient reuse of overlapping subgraphs and significantly reducing redundant computation.

References

[1] N. Prazeres, R. L. de C. Costa, L. Santos, and C. Rabadão, "Engineering the application of machine learning in an ids based on iot traffic flow," *Intelligent Systems with Applications*, vol. 17, p. 200189, 2023.

[2] S. Sriram, R. Vinayakumar, M. Alazab, and S. KP, "Network flow based iot botnet attack detection using deep learning," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 189–194, 2020.

[3] I. Ullah and Q. H. Mahmoud, "Design and development of rnn anomaly detection model for iot networks," *IEEE Access*, vol. 10, pp. 62722–62750, 2022.

[4] M. Kumar, C. Kim, Y. Son, S. K. Singh, and S. Kim, "Empowering cyberattack identification in ioht networks with neighborhood-component-based improvised long short-term memory," *IEEE Internet of Things Journal*, vol. 11, no. 9, pp. 16638–16646, 2024.

[5] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 550–559, PMLR, 10–15 Jul 2018.

[6] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018.

[7] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," 2020.

[8] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[9] S. Zawad, C. Li, Z. Yao, E. Zheng, Y. He, and F. Yan, "DySR: Adaptive super-resolution via algorithm and system co-design," in *The Eleventh International Conference on Learning Representations*, 2023.

[10] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[11] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *International Conference on Learning Representations*, 2020.

[12] V. Safronov, A. M. Mandalari, D. J. Dubois, D. Choffnes, and H. Haddadi, "Sunblock: Cloudless protection for iot systems," 2024.

[13] Google, "Colaboratory." https://research.google.com/colaboratory/, 2025. Accessed: 2025-05-30.

[14] M. Luay, S. Layeghy, S. Hosseininoorbin, M. Sarhan, N. Moustafa, and M. Portmann, "Temporal analysis of netflow datasets for network intrusion detection systems," 2025.

[15] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. T. H. d. Hartog, "Ton_iot: The role of heterogeneity and the need for standardization of features and attack types in iot network intrusion data sets," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 485–496, 2022.