Optimization of the D2D Topology Formation Using a Novel Two-Stage Deep ML Approach for 6G Mobile Networks

Iacovos Ioannou* Marios Raspopoulos‖, Prabagarane Nagaradjane‡, Christophoros Christophorou*, Ala' Khalifeh¶, Vasos Vassiliou*

* Department of Computer Science, University of Cyprus and CYENS - Centre of Excellence, Cyprus

‡ Department of ECE, Sri Sivasubramaniya Nadar College of Engineering Chennai, India

¶ German Jordanian University, Amman, Jordan

‖ INSPIRE Research Centre, University of Central Lancashire, Larnaca, Cyprus

***Abstract*—Optimizing device-to-device (D2D) topologies is pivotal for enhancing the performance and efficiency of 6G networks. This paper introduces a novel approach for forming optimal subnet trees within the 6G networks using BDIx agents and advanced Minimum-Weight Spanning Tree (MWST/MST) algorithms augmented by Graph Neural Networks (GNNs) and FeedForward Neural Networks (FFNN). Our solution aims to significantly boost network performance, particularly in high-demand scenarios such as urban areas, large-scale events, and remote locations. Our approach dynamically adapts to changing network conditions, user movements, and traffic patterns by minimizing power consumption and maximizing throughput. We implement various MWST algorithms, including Kruskal's, Prim's, and Boruvka's algorithms, and introduce a GNN model to predict edge weights combined with FFNNs to select parent nodes (called GNN-FFNN model), aiding in the construction of minimum-weight spanning trees (MWST). Additionally, a "weighted distance" metric is proposed to analyze network performance comprehensively. The proposed AI/ML-driven solution integrates BDIx agents with MWST algorithms, focusing on optimizing subnets under gNodeB in 6G networks, enhancing data transmission efficiency, reducing latency, and increasing throughput. This research contributes to developing scalable and flexible network management solutions suitable for diverse configurations and architectures.**

***Keywords*—Device-to-device (D2D) communication, 6G networks, Minimum-Weight Scanning Tree (MWST), Graph Neural Networks (GNNs), FedForward Neural Networks (FFNN), BDIx agents, network optimization, power consumption, throughput, and dynamic adaptation.**

I. Introduction

Optimizing D2D topologies is essential in the ever-changing telecommunications industry to improve the performance and efficiency of 6G networks. This research focuses on the issue of creating optimal subnet trees in these 6G networks. This is crucial for effectively managing the complex weighted tree structures needed for efficient network governance. Our proposal involves utilizing BDIx agents and sophisticated Minimum-Weight Spanning Tree (MWST) algorithms while employing Graph Neural Networks (GNNs) with FeedForward Neural Networks (FFNN) called GNN-FFNN. The objective of this research is to provide a robust and flexible solution to significantly improve network performance, particularly in situations with high demand like densely populated urban areas, large-scale events, and remote locations where maximizing coverage and connectivity is crucial [1], [2].

The primary challenge in optimizing D2D topologies is minimizing power consumption and maximizing throughput, such as sum rate, by selecting paths with the shortest distances and highest data rates. Traditional spanning tree methods often fail to adapt dynamically to changing network conditions, user movements, and traffic patterns, highlighting the need for a more robust and intelligent solution [3], [4]. Our research addresses these limitations by introducing a dynamic and adaptive service to optimize subnet trees in real time. Utilizing BDIx agents based on the Belief-Desire-Intention (BDI) framework augmented with machine learning, we provide a comprehensive depiction of the network's current state, including node positions and connection quality, which are critical for practical MWST algorithm application [5], [2]. This involves creating random trees and DAI framework-generated trees with BDIx agents, calculating data rates, and applying various MWST algorithms, including a GNN model to predict edge weights along with an FFNN to predict the parent of the investigated node. Continuing, this investigation evaluates the performance of the proposed approach along with the random BDIx agents and Kruskal's, Prim's, and Boruvka's algorithms. Power consumption and data rates are also calculated, introducing a "weighted distance" metric for comprehensive network performance analysis. This AI/ML-driven solution focuses on optimizing subnets in 6G networks under gNodeB, integrating BDIx agents with MWST algorithms using GNNs and FFNNs [6], [2].

The core purpose of our proposed strategy is to manage the complex weighted tree structures produced by BDIx agents, which is crucial for the closed-loop governance module. By employing innovative MWST algorithms, we aim to optimize network topology efficiently, considering key metrics like minimizing delay and maximizing throughput. Our solution's dynamic and flexible characteristics allow it to process new data from the localization submodule, adjusting to network condition fluctuations, user movement, and traffic patterns for continuous optimization. Designed for scalability and flexibil-

ity, our approach is compatible with various gNodeB configurations and 5G architectures, improving data transmission efficiency, reducing latency, and increasing throughput. It offers significant advantages in high-density urban areas, large-scale events, emergency situations, and remote areas with limited infrastructure [7], [8], [9]. Our strategy, utilizing a GNN with SAGEConv layers, first learns node embeddings and then uses these embeddings to predict parent-child relationships in the graph and with the use of FFNNs with the fully connected leayers to select the most appropriate parent of an investigated node in a tree, optimizing network topology dynamically and efficiently. The novelty of our approach lies in combining BDIx agents with GNN-FFNN-based MWST algorithms. This integration allows dynamic network configuration adjustments in response to real-time changes or improvements that can be enabled, ensuring peak network efficiency. The use of GNNs enhances the capability to handle complex graph structures, providing a scalable solution for various network sizes and complexities [10], [11] and the parent selection using FFNN helps the GNN on the identification of the MWST.

The contributions of this research are:

1) Leveraging the Distributed Artificial Intelligence (DAI) framework with machine learning to provide an adaptive solution for network management [12].
2) Implementing over the state-of-the-art MWST [1] algorithm based on two-stage Deep ML (GNN-FFNN) that ensures optimal path selection, minimizing latency and maximizing throughput.
3) The service adapts in real-time to changing conditions, user movements, and traffic patterns, ensuring consistent performance.
4) The solution is scalable and flexible, suitable for various network sizes and complexities.
5) Improvements in data transmission efficiency, reduce latency, increase throughput, and optimize resources [13], [9].

The rest of this article is arranged as follows: Section II provides a comprehensive discussion of the related work and background information that is relevant to our inquiry. The proposed system description is explained in detail in Section III. Section IV elaborates on the approach employed, the deep learning models utilized for the estimation, and the methodology used. The simulation results of the suggested system under different settings are presented and analyzed in Section V. Section VI discusses the results drawn from the research and outlines potential future directions.

II. Related Work and System Description

A. Related Work

The study in [14] introduces a high-altitude platform (HAP) assisted communication network for unmanned ship (USV) formations, using reinforcement learning and the whale optimization algorithm (WOA) to optimize network topology, resulting in improved data transfer rates, reduced D2D delays, and increased network throughput. Similarly, [15] proposes a modified Ring All-Reduce (MRAR) architecture to enhance federated learning (FL) communication efficiency using D2D wireless networks, with an Ant Colony Optimization-based algorithm optimizing the ring topology, significantly reducing transmission time and ensuring robust communication. In [16], graph-aware deep reinforcement learning (DRL) optimizes UAV swarm control via D2D communication, enhancing control accuracy and reducing latency through dynamic topology adjustments. [17] employs multi-agent reinforcement learning (MARL) to optimize mesh wireless network topologies dynamically, improving connectivity and throughput while reducing latency. Lastly, [18] presents a topology learning approach for decentralized, federated learning over unreliable D2D networks, reducing convergence time and communication overhead while maintaining high learning accuracy through dynamic network structure adjustments.

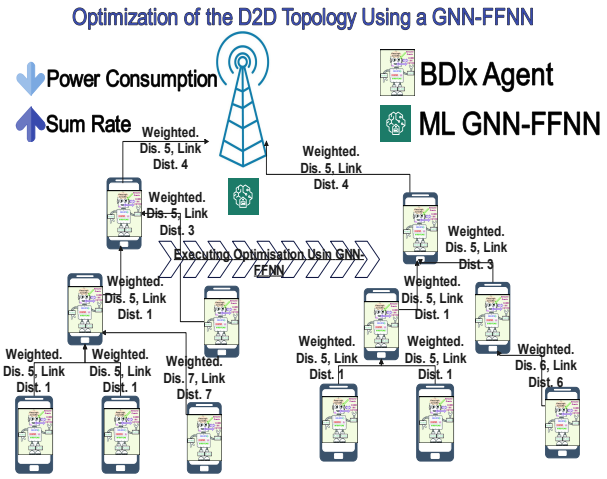


Figure 1: The System Architecture

B. Background Work

We discuss the background work necessary to understand the implementation of traditional Minimum Spanning Tree (MST) algorithms. These algorithms aim to find a subset of the edges that form a tree, including every vertex, where the total weight of all the edges in the tree is minimized. Here, we outline four classical algorithms and one ML approach: Kruskal's, Prim's, and Boruvka's approaches, along with the GNN approach, providing detailed explanations and their respective algorithms.

1) **Kruskal's Algorithm**: A greedy method sorting edges by weight and adding them to the MST without forming cycles until $V-1$ edges are included [8], [19].
2) **Prim's Algorithm**: Constructs the MST by starting with one vertex and repeatedly adding the smallest edge in terms of the metric examined connecting a vertex in the tree to one outside it [7], [20].
3) **Boruvka's Algorithm**: Treats each vertex as a separate component, merging the closest components iteratively until a single component remains [21], [22].
4) **GNN Approach**: Uses SAmple and aggreGatE Convolution (SAGEConv) layers to aggregate information from node neighbourhoods, generating embeddings for

[1] MWST algorithms are called like this because we use the weighted metric. They are the same as the MST.

tasks like node classification and MST prediction [13], [23]. The architecture involves a sequence of SAGEConv layers followed by ReLU activations. Each layer aggregates features from neighboring nodes to update node embeddings, capturing local graph structure and node attributes. The final node embeddings are then used for various tasks, such as predicting the weights of edges in the network for MWST construction.

5) **FFNN Approach**: Utilizes a FFNN architecture with fully connected layers and ReLU activations [24]. The architecture consists of an input layer, multiple hidden layers, and an output layer. Each layer is fully connected to the next, with ReLU activations applied to introduce non-linearity. The FFNN processes the embeddings generated by the GNN to predict the parent nodes in the MWST, effectively determining the optimal tree structure by evaluating potential parent-child relationships and selecting the most efficient configuration based on learned patterns.

These algorithms have different strengths and weaknesses. Kruskal's and Prim's are general-purpose algorithms, while Boruvka's algorithm excels in parallel computing. The GNN-FFNN combined approach leverages learned embeddings for potentially improved MST predictions.

III. System Description

This section outlines the problem and system components. The proposed system comprises a base station (BS), User Equipment (UE) from multiple users that are forming tree-style D2D communication subnetworks, and a BS controller responsible for optimizing the network tree as shown in Figure 1. The BS coordinates the network, while UEs participate in D2D communications. The BS controller ensures efficient network management and performance. BDIx agents, based on the DAI framework enhanced with machine learning, autonomously form subnetworks under the BS using a transmission selection algorithm that employs the Weighted Data Rate (WDR) metric. This allows BDIx agents to dynamically select transmission modes and establish efficient subnets by considering data rates and UE positions. These agents provide a comprehensive view of the network's current state, including node positions and connection quality, aiding the application of MWST algorithms for optimizing network topology. Initially, BDIx agents form a subnetwork under the BS. We aim to optimize this subnetwork to enhance the existing network's performance (i.e., sum rate and power consumption). When traffic and user demand increase in a specific 5G subnetwork, the telecom operator can activate the BDIx agents to form a D2D communication network targeting them, increasing data rates, reducing power consumption, and supporting the required bandwidth.

IV. Deep Learning Aided Topology Optimization Methodology

In this section, we provide the introduction of the weighted distance, the methodology, and the steps that are used in the research. So, this study focuses on creating and analyzing Minimum Spanning Trees (MST) to minimize the maximum weighted distance in graphs representing random trees and trees generated by BDIx agents with coordinates. We use traditional MST algorithms and a GNN approach to predict parent-child relationships in the graph, and the FFNN is used to select the parent of each node. This section details our methodology, including data generation, algorithmic steps, training process, hyperparameter search, evaluation, and visualization. Our methodology involves several key steps. First, we create a structure to track the results for each model, organizing various metrics such as data rates and power consumption values in a dictionary. This organization ensures that all necessary data is easily accessible. We then iterate through different node sizes, generating sample data for each size to assess model performance as the network scales. The current node size being tested is recorded to maintain clear conditions for each result set. Next, we use Kruskal's algorithm to compute the MST for the generated data, finding the subset of edges that connects all nodes with the minimum total edge weighted distance. The total weighted distance, data rate, and power consumption for the MST are calculated and recorded, serving as benchmark data for training GNN-FFNN-based models. We then retrieve the necessary parameters for each model, configure them, and execute them to generate predictions and construct trees. Performance metrics for these predicted trees, such as total weight, data rate, and power consumption, are calculated and recorded to compare each model's effectiveness against the benchmark.

Finally, we compile all recorded results into a comprehensive dataset containing performance metrics for each model across different node sizes. This dataset is returned for further analysis and comparison, ensuring that all evaluation data is systematically organized and ready for detailed examination.

A. Formulation of Weighted Distance

In order to calculate the MST towards the root node, referred to as the BS, we need to introduce a new metric, the weighted distance. The weighted distance is defined as the maximum link distance of an edge on a path towards the root. Specifically, for any two nodes u and v, the weighted distance (physical distance in meters) $w(u,v)$ is given by the following formulation (Eq. 1):

$$w(u,v) = \max_{e \in P(u,v)} \{d(e)\} \tag{1}$$

where $P(u,v)$ represents the path between nodes u and v, and $d(e)$ is the distance of edge e on this path (as shown in 1). This metric allows us to create a complete graph where the edge weights reflect the maximum link distance on a path toward the root. This setup enables a comprehensive comparison of the performance of different MST algorithms and our GNN-FFNN-based model and also provides us with the tool to calculate the minimum spanning tree that its links have the minimum weighted distance towards the BS. The weighted distance metric was chosen because it allows for a comprehensive comparison of MST algorithms and the GNN-FFNN-based model by reflecting the maximum link distance on a path towards the root, ensuring the minimum spanning tree has the minimum weighted distance towards the base

station. This setup is crucial for effectively optimizing device-to-device communication topologies.

B. Methodology Steps

1) Data Creation: The data creation process involves generating synthetic data using random trees and complete graphs, focusing on calculating maximum distance paths and setting edge weights. Start by creating a random tree with a specified number of nodes and assign coordinates to each node for spatial representation. Calculate and assign the weighted distance between connected nodes as the edge's weight, along with the distances of each edge. Designate the root node (node 0) as the BS and compute the maximum distance paths from this root to all other nodes, utilizing the previous weighted distance calculations. Generate a complete mesh graph where every pair of nodes is connected, representing all possible direct connections. Assign the same coordinates to each node in the complete graph as in the tree to maintain spatial consistency. Calculate the distance for each edge and set its weight to the maximum of the pre-calculated maximum distances of the connected nodes. Kruskal's algorithm is used to construct a minimum spanning tree (MST) from the complete graph, determining the minimum distance for each edge in the MST. Finally, convert the complete graph and the MST into an appropriate format, likely an array, for further analysis or model training. We execute the same procedure multiple times in order to generate a large dataset on which our model can be trained.

2) The Training Features: The dataset used for training the GNN model consists of several key fields as outlined in Table I. These fields include the node features, which are the coordinates of each node in the graph; the edge index, which defines the connections between nodes; and the edge attributes, which are the weights of the edges based on node distances. Additionally, the dataset includes parent indices, representing the parent-child relationships in the Minimum Spanning Tree (MST), which are used as labels for training the model.

Table I: Fields of the Dataset Used for Training the GNN Model

Dataset Field	**Description**	**Tensor Shape**
Node Features (pos)	Coordinates (x, y) of each node	(`num_nodes`, 2)
Edge Index (edge_index)	Indices of nodes forming each edge	(2, `num_edges`)
Edge Attributes (edge_attr)	Weights of edges based on distance	(`num_edges`, 1)
Parent Indices (parent_indices)	Parent node index in the MST	(`num_nodes`)

For the training of the FFNN, we use the potential parent nodes given from the GNN. So, the FFNN is trained along with the GNN.

3) Hyperparameter Search: To optimize the models' performance, we conduct a hyperparameter search using the Optuna library over a predefined grid of parameters, selecting the best combination based on validation loss [25]. The search process involves exploring various combinations of hyperparameters to identify the best set for a given model. This is achieved through a systematic search over a predefined grid of hyperparameter values, using K-Fold cross-validation to evaluate each combination. The hyperparameter search process begins with the initialization step. Next, the grid search step iterates over all combinations of the hyperparameters, and the process continues until, at the end, the best hyperparameters of the model are identified.

In the models training and evaluation step, the models are trained and evaluated using the function `TrainAndEvaluate` with K-Fold cross-validation for each combination of hyperparameters. The average validation loss, *avg_val_loss*, is computed from the validation losses obtained from the K-Folds using the following formulation (Eq. 2):

$$avg\_val\_loss = \frac{1}{k}\sum_{i=1}^{k} val\_loss_i \tag{2}$$

During the best parameters update step, if $avg\_val\_loss < best\_score$, the variable *best_score* is updated to *avg_val_loss*. The variable *best_params* is then set to the current combination of hyperparameters, which includes *hidden_channels1*, *hidden_channels2*, and *output_dim*. Finally, after evaluating all combinations, the best parameters and the best score are returned. In summary, the hyperparameter search process systematically explores various combinations of hyperparameters using K-Fold cross-validation to identify the best set that minimizes the validation loss. This approach ensures the selection of optimal hyperparameters for the given model.

4) Validation using K-Fold Cross-Validation Approach: The training process involves K-Fold cross-validation to ensure robust evaluation. The models are trained on the dataset using backpropagation and the Adam optimizer. Evaluation metrics include mean absolute error (MAE), mean squared error (MSE), root mean square error (RMSE), and R^2 score [26]. The detailed training procedure is as follows:

First, the number of classes is determined by finding the maximum parent index in the training data. The K-Fold cross-validation is initialized with the specified number of splits, and an empty list is created to store the results from each fold. For each fold in the K-Fold split, the training and testing subsets are created from the training data list based on the indices provided by the K-Fold split. For each batch in the training loader, the optimizer gradients are zeroed, and the batch data is loaded. The model computes node embeddings from the batch data, and the parent predictor produces output from these node embeddings. The loss is computed using the criterion on the output and the batch's parent indices. The loss is then backpropagated, and the optimizer steps to update the model parameters. Finally, the training loss for the batch is added to the total training loss. After processing all batches, the average training loss is computed. The model is evaluated on the test loader using the defined evaluation metrics, and the results are appended to the list of fold results. Finally, the function returns the accumulated results from all folds, and the best split of training and test percentage is selected, which is 80% to 20%.

5) Model Implementation: Graph Neural Networks (GNNs) have emerged as a powerful tool for learning graph-

structured data along with a feedforward neural network for parent prediction.

Algorithm 1 GNN Model

function GNNMODEL(hidden_channels1, hidden_channels2, output_dim)
    $self.conv1 \leftarrow$ SAGECONV(2, hidden_channels1)
    $self.conv2 \leftarrow$ SAGECONV(hidden_channels1, hidden_channels2)
    $self.conv3 \leftarrow$ SAGECONV(hidden_channels2, output_dim)
    $self.linear \leftarrow$ LINEAR(output_dim, output_dim)
end function
function FORWARD(data)
    $x, edge\_index \leftarrow data.x, data.edge\_index$
    $x \leftarrow$ RELU(self.conv1(x, edge_index))
    $x \leftarrow$ RELU(self.conv2(x, edge_index))
    $x \leftarrow self.conv3(x, edge\_index)$
    $x \leftarrow self.linear(x)$
    return x
end function

Specifically, we implemented a GNN using SAGEConv layers to learn node embeddings for predicting MST-related properties [13]. The SAGEConv layer, or GraphSAGE convolution, aggregates features from a node's local neighborhood to generate its embedding. This method allows for efficient computation and scalability to large graphs. The key steps in our Graph Neural Network (GNN) implementation involve a specific process. The process begins by initializing the SAGEConv layers, which are essential for aggregating information from neighboring nodes. Next, forward propagation is performed through the network to generate node embeddings, capturing the structural and feature-based information of the nodes. Finally, these embeddings are utilized to predict properties related to the minimum spanning tree (MST), leveraging the learned representations to infer relevant MST characteristics.

Algorithm 2 Parent Predictor Model

function PARENTPREDICTOR(input_dim, hidden_dim, num_classes)
    $self.lin1 \leftarrow$ LINEAR(input_dim, hidden_dim)
    $self.lin2 \leftarrow$ LINEAR(hidden_dim, hidden_dim)
    $self.output \leftarrow$ LINEAR(hidden_dim, num_classes)
end function
function FORWARD(x)
    $x \leftarrow$ RELU(self.lin1(x))
    $x \leftarrow$ RELU(self.lin2(x))
    return $self.output(x)$
end function

In the process, we utilized the Parent Predictor model as shown in Algorithm 2, which is an integral component within the graph neural network (GNN) framework designed to predict parent nodes for each node in a graph, essential for tasks like constructing minimum spanning trees (MSTs). It operates alongside an Enhanced GNN model, which processes node features and edge connections to generate embeddings. The Parent Predictor, a feedforward neural network with fully connected layers and ReLU activations, uses these embeddings to predict parent nodes by outputting logits for each possible parent. During training, both the GNN and the Parent Predictor are optimized using a cross-entropy loss function to minimize the discrepancy between predicted and actual parent nodes. This combined approach leverages the GNN for rich feature extraction and the Parent Predictor for hierarchical relationship prediction, which is crucial for evaluating MST properties and ensuring efficient and accurate graph representations. The parent prediction predicts the correct parent of a node base on the list of parents that the GNN model will provide.

Algorithm 3 Evaluate Models

function EVALUATEMODELS(models, best_params, num_nodes_range)
    $test\_results \leftarrow$ dictionary of result lists
    for all $model\_class \in models$ **do**
        $model\_name \leftarrow model\_class.\_\_name\_\_$
        Initialize $test\_results$ lists for $model\_name$ data rate and power consumption
    end for
    for all $num\_nodes \in num\_nodes\_range$ **do**
        $data, initial\_BDIx\_tree, full\_graph, kruskal\_mst \leftarrow$ CREATESAMPLEDATA(num_nodes)
        $test\_results['num\_nodes'].append(num\_nodes)$
        $mst, kruskal\_weight \leftarrow$ KRUSKALMST(full_graph)
        $kruskal\_data\_rate, kruskal\_power\_loss \leftarrow$ CALCULATEMSTMETRICS(full_graph, kruskal_mst)
        Append $kruskal\_weight$, $kruskal\_data\_rate$, and $kruskal\_power\_loss$ to $test\_results$
        for all $model\_class \in models$ **do**
            $model\_name \leftarrow model\_class.\_\_name\_\_$ which is GNN
            Retrieve model-specific parameters and instantiate model
            Construct $predicted\_tree$ from $predicted\_parents$ with the use of FFNN
        end for
    end for
    return $test\_results$
end function

6) Train and Evaluation: The GNN implementation using SAGEConv layers takes a data object as input, which contains node (network device) features and edge (connection) indices. A sequential model is created, and it consists of a SAGEConv layer that takes the input channels and transforms them into hidden channels, followed by a ReLU activation function. Another SAGEConv layer takes the hidden channels and transforms them into output channels. An optimizer is created using the Adam optimization algorithm, with the model's parameters and a learning rate of 0.01. The training process runs for a specified number of epochs. During training, the model is set to training mode, and the gradients of the

optimizer are zeroed. Forward propagation is performed on the input data to generate node embeddings, using the node features and edge indices as inputs to the model. The loss is calculated using a specified loss function, which compares the model's output with the target labels. Backward propagation is performed to compute the gradients, and the optimizer steps to update the model parameters. After completing the training epochs, the function returns the trained model. This model learns node embeddings and uses a fully connected neural network to predict parent-child relationships. The architecture of the GNN model is described in Algorithm 1. The `GNNModel` function is called to initialize the model with the specified layers and dimensions. The `Forward` function is called during the forward pass to compute the node embeddings. Thus, the `GNNModel` function is called during model initialization to set up the convolutional layers (`conv1`, `conv2`, `conv3`) and the linear layer (`linear`). The `Forward` function is executed during each forward pass through the network, where it takes the input data, applies the convolutional layers with ReLU activations, and then applies the final linear transformation to produce the output node embeddings. To predict parent nodes, we implemented a Parent Predictor model (Algorithm 2). The `ParentPredictor` function initializes the model with the specified dimensions for the input, hidden layers, and output. The `Forward` function is called to compute the predictions for the parent nodes. The `ParentPredictor` function is called during model initialization to set up the linear layers (`lin1`, `lin2`, `output`). The `Forward` function is executed during each forward pass, where it takes the input node embeddings, applies the linear transformations with ReLU activations, and then produces the final output, which is the predicted parent indices for each node. These models work together first to learn the node embeddings using the GNN with `SAGEConv` layers and then use these embeddings to predict the parent-child relationships in the graph. The execution steps are as follows: First, the model is initialized by defining a sequential model with `SAGEConv` layers, where the first `SAGEConv` layer transforms the input features into hidden representations, followed by a `ReLU` activation function to introduce non-linearity, and the second `SAGEConv` layer transforms the hidden representations into output features. Next, an `Adam` optimizer is initialized with the model's parameters and a learning rate of 0.01, which will be used to update the model parameters during training. The training loop iterates over a predefined number of epochs, wherein each epoch, the model is set to training mode to ensure correct layer behavior, the optimizer's gradients are reset to zero to prevent accumulation from previous iterations, forward propagation is performed to compute the output embeddings from the input node features and edge indices, the loss is calculated using a loss function that measures the difference between the model's output and the true labels, backward propagation is performed to compute the gradients, and the model's parameters are updated based on the gradients using the optimizer. After completing all training epochs, the trained GNN and FFNN models are returned. These trained models can then be used to predict MST-related properties based on the node embeddings generated during forward propagation. We evaluate the performance of our models on a range of node sizes, comparing the GNN-FFNN-based approach with traditional MST algorithms in terms of total weighted distance, data rate, and power consumption. The detailed evaluation procedure is structured to ensure a comprehensive assessment of each model's performance across different node sizes. To thoroughly evaluate the performance of our models on varying node sizes, we compared the GNN-FFNN-based approach with traditional MST algorithms. The focus was on metrics such as total weighted distance, data rate, and power consumption. The evaluation process systematically tests each model across a range of node sizes and compares their performance against the Kruskal MST algorithm, serving as a benchmark. The entire evaluation procedure is encapsulated in Algorithm 3. This algorithm outlines the step-by-step process for evaluating the models, ensuring a systematic and repeatable method for performance assessment. Our proposed methodology combines traditional and modern machine learning techniques to tackle the optimization problem of MSTs, leveraging both approaches' strengths for comprehensive analysis and comparison. The two-stage Deep ML model from now and on it will be called GNN-FFNN.

V. Simulation Results and Analysis

This section provides a description of the metrics that are used in the examination. It also evaluates the performance of the proposed deep learning-enhanced optimization of the D2D communication model with traditional approaches using specific network metrics. Moreover, it examines the impact of NN and GNN-FFNN estimation over the ML metrics (i.e., Train Loss, Validation Loss, MAE, MSE, RMSE, R^2, adj R^2[22]) measured in terms of weighted distance in various scenarios. The training and testing sets consist of 8000 and 2000 samples, respectively. The simulation parameters are presented in Table II.

Table II: Simulation Parameters

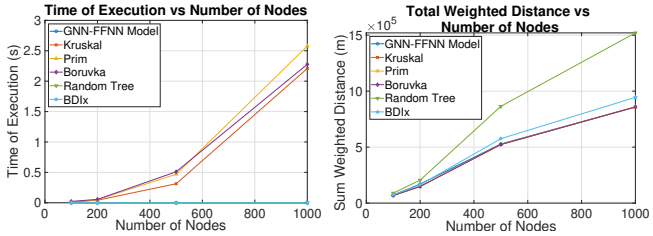
Parameter	Value
Frequency	2.4 GHz
Transmit Power	20 dBm
Gain	2 dB
Noise Figure	10 dB
Bandwidth using WiFi Direct	1 MHz
Path Loss Exponent	2
Noise Floor Level	-174 dBm/Hz

A. Results and Analysis Regarding Network Metrics

This section presents an in-depth analysis of the network metrics for optimizing D2D communication topologies using

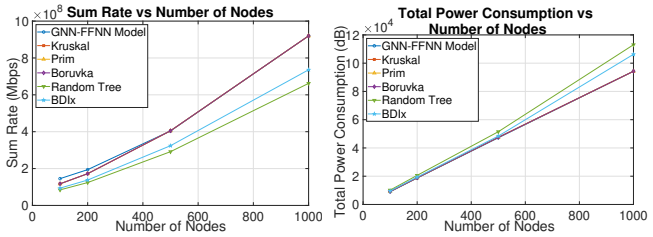
[2]Train Loss: Measures how well the model fits the training data. A lower value indicates a better fit. Validation Loss: Evaluates model performance on unseen validation data. Lower values signify better performance. MAE (Mean Absolute Error): Averages the absolute differences between predicted and actual values. Indicates model accuracy in terms of average error. Lower values are better. MSE (Mean Squared Error): Averages the squared differences between predicted and actual values. Penalizes larger errors more than smaller ones. Lower values are preferable. RMSE (Root Mean Squared Error): Square root of the MSE. Maintains the same unit as the target variable. R^2 (R-squared): Represents the proportion of variance in the target variable explained by the model. Ranges from 0 to 1. Higher values indicate better explanatory power. Adj R^2 (Adjusted R-squared): Adjusts the R^2 value for the number of predictors in the model [26].

various algorithms, including a novel GNN-FFNN-based approach. The metrics include time of execution, total weighted distance, data rate, and total power consumption. The results are based on the number of nodes in the network.



(a) Time of Execution vs Number of Nodes

(b) Total Weighted Distance vs Number of Nodes



(c) Data Rate vs Number of Nodes

(d) Total power consumption vs Number of Nodes

Figure 2: Comparison of network metrics for different algorithms as the number of nodes increases.

1) Time of Execution: Figure 2a shows the time of execution for different algorithms as the number of nodes increases. The GNN-FFNN model consistently demonstrates lower execution times compared to most other algorithms, particularly for larger networks. This efficiency is crucial for real-time applications where rapid computation is necessary. The GNN-FFNN model's ability to quickly compute optimal topologies stems from its deep learning-based approach, which effectively generalizes from training data to make fast predictions during inference. Kruskal's algorithm shows a similar trend but slightly higher execution times, making it another viable option for real-time applications. Prim's algorithm, however, has the highest execution time among all the compared algorithms, indicating it may not be suitable for real-time applications in large networks. Both Boruvka and Random Tree algorithms have higher execution times than the GNN-FFNN model but are still lower than Prim's. The BDIx algorithm shows competitive execution times, although not as low as the GNN-FFNN model.

2) Total Weighted Distance: Figure 2b illustrates the total weighted distance for different algorithms. The GNN-FFNN model maintains a lower total weighted distance compared to the Random Tree algorithm, indicating its effectiveness in optimizing the paths within the network. This metric is crucial for minimizing latency and ensuring efficient data routing. The GNN-FFNN model's ability to learn from graph structures allows it to make intelligent decisions about path optimization, leading to lower weighted distances. Kruskal and Prim algorithms perform comparably well, with slightly higher total weighted distances than the GNN-FFNN model. Boruvka also performs well but is not as consistent as the GNN-FFNN model. The Random Tree algorithm shows significantly higher weighted distances, indicating poor path optimization. The BDIx algorithm performs better than Random Tree but not as well as the GNN-FFNN model.

3) Sum Rate: Figure 2c presents the data rate for different algorithms. The GNN-FFNN model achieves a high data rate, close to that of Boruvka's algorithm, and significantly better than the Random Tree algorithm. This high data rate indicates the GNN-FFNN model's efficiency in maximizing network throughput, which is essential for supporting high-bandwidth applications. The model's capability to understand and optimize the network's data flow patterns contributes to this high performance. Boruvka's algorithm achieves a slightly higher data rate than the GNN-FFNN model, but both are very close in performance. Kruskal and Prim algorithms perform well but have lower data rates compared to the GNN-FFNN model. The Random Tree algorithm, on the other hand, shows a lower data rate, reflecting its inefficiency in optimizing data transmission paths. The BDIx algorithm performs comparably to the GNN-FFNN model, indicating efficient data routing.

4) Total power consumption: Figure 2d depicts the total power consumption for different algorithms. The GNN-FFNN model exhibits lower power consumption compared to the Random Tree algorithm, demonstrating its energy efficiency. Minimizing power consumption is critical for extending the lifespan of network devices and reducing operational costs. The GNN-FFNN model's optimization process takes into account not only the network's connectivity but also its power consumption patterns, resulting in lower total power consumption. Kruskal and Prim algorithms perform comparably to the GNN-FFNN model in terms of power consumption, indicating similar levels of energy efficiency. Boruvka has a slightly higher power consumption but remains efficient. The Random Tree algorithm shows the highest power consumption, indicating inefficiency in power management. The BDIx algorithm performs well, with power consumption close to the GNN-FFNN model, highlighting its effectiveness in minimizing energy consumption.

5) Detailed Comparison with Traditional Algorithms and GNN-FFNN: The GNN-FFNN model's performance is compared with traditional algorithms like Kruskal, Prim, Boruvka, and Random Tree. The GNN-FFNN model excels in execution time, making it ideal for large-scale real-time applications, while Kruskal's algorithm follows with slightly higher times. Prim's algorithm, with the highest execution time, is less suitable for real-time use. Both Boruvka and Random Tree algorithms have higher execution times than the GNN-FFNN model but are still lower than Prim's, with the BDIx algorithm being competitive but not as efficient as the GNN-FFNN model. In terms of total weighted distance, the GNN-FFNN model outperforms the Random Tree algorithm, indicating better path optimization, with Kruskal and Prim performing slightly worse but comparably well. Boruvka is less consistent, and the Random Tree algorithm shows poor optimization, while the BDIx algorithm is better than Random Tree but not as good as the GNN-FFNN model. Regarding data rate, the GNN-FFNN model achieves high data rates similar to Boruvka's, indicating efficient throughput, with

Kruskal and Prim having lower rates. The Random Tree algorithm shows inefficiency in data transmission, while the BDIx algorithm performs comparably to the GNN-FFNN model. In terms of power consumption, the GNN-FFNN model is energy-efficient compared to the Random Tree algorithm, with Kruskal and Prim performing similarly well. Boruvka has slightly higher consumption but remains efficient, while the Random Tree algorithm is the least efficient. The BDIx algorithm shows close performance to the GNN-FFNN model, indicating effectiveness in minimizing energy use.

B. Results and Analysis Regarding ML Metrics

The dataset provided includes various metrics for evaluating a machine learning model. Here is an in-depth analysis of the results:

Table III: Model Evaluation Metrics

Train Loss	**Val Loss**	**MAE**	**MSE**	**RMSE**	R^2	**Adj** R^2
0.0338	0.0044	0.0143	0.9855	0.9928	0.9661	0.9661

According to Table III, the training loss is 0.0338, while the validation loss is significantly lower at 0.0044. This indicates that the model performs better on the validation dataset than the training dataset. The MAE is 0.0143, which shows the very low average magnitude of errors in a set of predictions without considering their direction. The MSE is 0.9855, indicating relatively low average squared errors. The RMSE of 0.9928, derived from the MSE, indicates the model's error magnitude is just below 1. The R^2 value is 0.9661, indicating that 96.6% of the variance in the dependent variable is predictable from the independent variables. The adjusted R^2 value is nearly identical at 0.9661, reinforcing the model's high explanatory power. The model exhibits excellent performance on both the training and validation sets, as indicated by the low loss values and high R^2 values. The error metrics (MAE, MSE, RMSE) suggest that the model's predictions have very low errors. The high R^2 and adjusted R^2 values indicate that the model explains a significant portion of the variance in the target variable, showcasing its robustness and predictive power.

VI. Conclusions and Future Work

This paper evaluates a deep learning-enhanced optimization of the D2D communication model against traditional methods. Results indicate that the GNN-FFNN model, along with Kruskal and BDIx algorithms, generally outperforms Prim, Boruvka, and Random Tree algorithms across various metrics. The GNN-FFNN model and Kruskal algorithm offer balanced performance with low execution times, optimized weighted distances, high data rates, and reduced power consumption, making them ideal for real-time applications in large D2D networks. The GNN-FFNN model achieves the lowest execution times, making it highly efficient for real-time applications. It also excels in path optimization, significantly reducing latency and ensuring efficient data routing. The GNN-FFNN model consistently achieves high data rates, demonstrating its ability to maximize network throughput, which is essential for high-bandwidth applications. Additionally, the GNN-FFNN model shows lower power consumption compared to other algorithms, emphasizing its energy efficiency and suitability for large-scale deployments where minimizing power consumption is critical. These findings suggest that the GNN-FFNN model is highly effective and efficient for modern communication networks requiring real-time data processing and energy efficiency. Its promising performance indicates that it can be a key component in future D2D communication systems. Overall, GNNs are employed to enhance the formation of optimal D2D communication topologies within 6G networks. Specifically, GNNs are used to predict edge weights in the network, aiding in the construction of MWST. These predictions, combined with FFNNs to select parent nodes, dynamically adapt to changing network conditions, optimize power consumption, and maximize data throughput, thereby improving overall network performance.

Future research should focus on several areas to further enhance the capabilities of the GNN-FFNN model. Scalability studies are needed to assess its performance in larger network scenarios, ensuring robustness and reliability. Real-world deployments in D2D communication systems will help validate its practical applicability and effectiveness. Investigating hybrid models that combine the GNN-FFNN with other techniques or traditional algorithms could lead to even better performance and efficiency. Additionally, future work should explore adaptive methods to dynamically adjust to network topology changes, develop techniques to handle environmental factors such as interference and mobility, conduct extensive testing in diverse real-world scenarios to ensure broad applicability and resilience and examine the integration of additional performance metrics like latency, packet delivery ratio, and network robustness under different failure scenarios to provide a more comprehensive evaluation of the model's effectiveness.

Acknowledgement

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 739578, the ADROIT6G project of the SNS-JU under Grant Agreement No. 101095363, and the Government of the Republic of Cyprus through the Deputy Ministry of Research, Innovation and Digital Policy.

References

[1] M. Eisen and A. Ribeiro, "Optimal Wireless Resource Allocation with Random Edge Graph Neural Networks," *IEEE Transactions on Signal Processing*, vol. 68, no. 1, pp. 2977–2991, 2020.

[2] I. Ioannou, V. Vassiliou, C. Christophorou, and A. Pitsillides, "Distributed Artificial Intelligence Solution for D2D Communication in 5G Networks," *IEEE Systems Journal*, vol. 14, no. 3, pp. 4232–4241, sep 2020. [Online]. Available: http://arxiv.org/abs/2001.05232https://ieeexplore.ieee.org/document/9080591/

[3] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *Journal of Big Data*, vol. 6, no. 1, 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[4] P. Tam, S. Ros, I. Song, S. Kang, and S. Kim, "A Survey of Intelligent End-to-End Networking Solutions: Integrating Graph Neural Networks and Deep Reinforcement Learning Approaches," *Electronics (Switzerland)*, vol. 13, no. 5, pp. 345–359, 2024.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[6] J. L. Nave, "Z–from theory to practice." *Management world*, vol. 12, no. 4, pp. 10–12, 1983.

[7] R. C. Prim, "Shortest Connection Networks And Some Generalizations," *Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[8] J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem (1956)," *Ideas That Created the Future*, vol. 7, no. 1, pp. 179–182, 2021.

[9] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[10] N. Shabani, J. Wu, A. Beheshti, Q. Z. Sheng, J. Foo, V. Haghighi, A. Hanif, and M. Shahabikargar, "A Comprehensive Survey on Graph Summarization with Graph Neural Networks," *IEEE Transactions on Artificial Intelligence*, vol. 31, no. 11, pp. 4–24, 2024.

[11] K. Xu, S. Jegelka, W. Hu, and J. Leskovec, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[12] V. Morreale, S. Bonura, G. Francaviglia, M. Cossentino, and S. Gaglio, "PRACTIONIST: a New Framework for BDI Agents," R&D Laboratory - Engineering Ingegneria Informatica S.p.A, Tech. Rep., 2005.

[13] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, vol. 2017-December, 2017, pp. 1025–1035.

[14] H. Cao, T. Yang, Z. Yin, X. Sun, and D. Li, "Topological optimization algorithm for HAP assisted multi-unmanned ships communication," *IEEE Vehicular Technology Conference*, vol. 2020-November, pp. 1–10, 2020.

[15] Z. Xu, W. Tian, Y. Liu, W. Ning, and J. Wu, "A Ring Topology-Based Communication-Efficient Scheme for D2D Wireless Federated Learning," *Proceedings - IEEE Global Communications Conference, GLOBECOM*, pp. 2820–2825, 2023.

[16] Y. Su, H. Zhou, and Y. Deng, "D2D-Based Cellular-Connected UAV Swarm Control Optimization via Graph-Aware DRL," *Proceedings - IEEE Global Communications Conference, GLOBECOM*, pp. 1326–1331, 2022.

[17] W. Sun, Q. Lv, Y. Xiao, Z. Liu, Q. Tang, Q. Li, and D. M. Mu, "Multi-Agent Reinforcement Learning for Dynamic Topology Optimization of Mesh Wireless Networks," *IEEE Transactions on Wireless Communications*, pp. 1–5, 2024.

[18] Z. Wu, Z. Xu, D. Zeng, J. Li, and J. Liu, "Topology Learning for Heterogeneous Decentralized Federated Learning Over Unreliable D2D Networks," *IEEE Transactions on Vehicular Technology*, pp. 1–8, 2024.

[19] S. Radhakrishnan, D. Kolippakkam, and V. S. Mathura, *Introduction to algorithms*. MIT press, 2007.

[20] R. E. Tarjan, *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[21] O. Boruvka and O. Jistem, "Problem minimalnim," *Praca Moravske Prirodovedecke. Spolecnosti*, vol. 3, pp. 37–59, 1926.

[22] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs," *Combinatorica*, vol. 6, no. 2, pp. 109–122, 1986.

[23] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

[24] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *Ieee Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[25] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, pp. 2623–2631.

[26] I. Ioannou, M. Savva, M. Raspopoulos, C. Christophorou, and V. Vassiliou, "Revolutionising IoT Network Security By Assessing ML Localisation Techniques Against Jamming Attacks," in *MedComNet 2024*, 2024.