Optimizing Energy Efficiency for Edge Caching for IIoT networks

Jovian Nursan Ng
U2020037L

Academic Year 2023-24
Submitted in Partial Fulfilment of the Requirements for the Degree of Bachelor of Engineering (Computer Science)
School of Computer Science & Engineering
Nanyang Technological University, Singapore

Supervisor: Assoc Prof A S Madhukumar
Co-Supervisor: Ernest Tan, Ritabrata Maiti

Abstract

The Internet of Things (IoT) is a dynamic global network infrastructure, where devices and sensors are interconnected, enabling information sharing and collective decision-making capabilities between the devices in the network [1]. IIoT focuses on improving industrial process productivity by interconnecting industrial devices such as AMRs and sensors to infrastructures such as BS and cloud servers. This facilitates the data exchange and analysis between devices within the IIoT network, as shown in Figure 1 below.

It requires a significant amount of energy consumption to maintain the transmissions within the IIoT network. As such, this project aims to propose a framework, mainly machine learning models, to optimize the energy EE of the system with the aim of minimizing the cost of maintaining the system. The models will be evaluated using the benchmark method as a baseline. Evaluation criteria include prediction accuracy and time taken compared to the benchmark method. This evaluation aims to predict optimal parameters for achieving optimal EE. The proposed machine learning models are shown to be able to predict around 800 times faster while only losing around 3-4% in terms of prediction accuracy compared to the benchmark method.

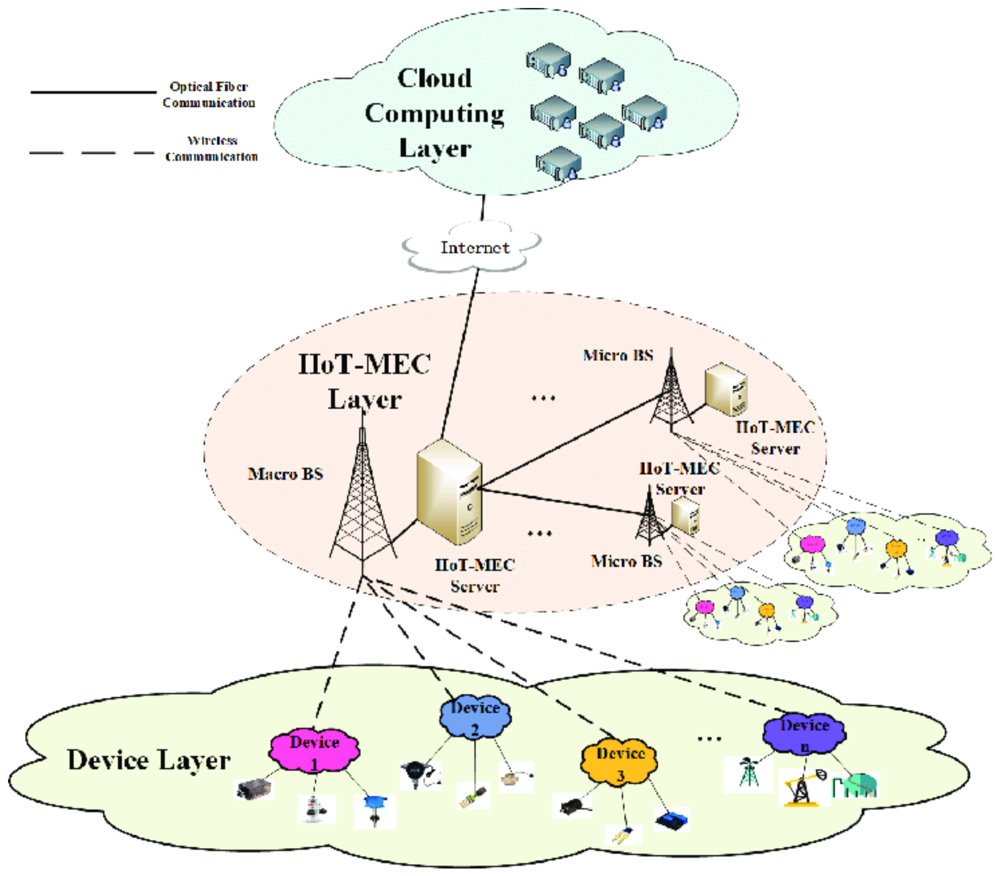


Figure 1: IIoT and MEC structure. Source : [5]

Acknowledgement

First and foremost I would like to thank my supervisors for their constant help and support throughout my Final Year Project journey.

I would like to thank Dr. Ernest Tan from Advanced Remanufacturing and Technology Centre (ARTC) for the constant guidance and knowledge sharing since the start of my FYP journey.

I would like to also thank Ritabrata Maiti for always spending the time helping me understand the concepts behind this project and also guiding me towards the objective of my FYP project.

Last but not least I would like to thank Assoc. Prof. A S Madhukumar for the consistent and constructive feedback you have given me until now for my own self-improvement.

Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

IIoT	Industrial Internet-of-Things
AMR	Autonomous Mobile Robot
BS	Base Station
MBS	Macro Base Station
SBS	Small Base Station
CBAS	Cache-Based Association Strategy
IAMR	Interfering Autonomous Mobile Robot
EE	Energy Efficiency
DL	Deep Learning

1. Introduction

The IIoT market in Singapore is rising through the years, with the IIoT market revenue increasing from US$ 0.42 billion in 2018 to US$ 1.13 billion in 2023. The IIoT market revenue is expected to rise to US$ 1.33 billion by the end of 2024 [2]. IIoT would be an essential part of Singapore's Smart Nation Program, which aims to enrich the lives of the citizens of Singapore, enabled seamlessly by technology. Businesses can also utilize these technologies to increase the productivity and efficiency of their processes as well as explore new opportunities to leverage on in the digital economy [3]. For example, Singapore furniture brand Star Living built their new warehouse which utilizes IIoT in Sungei Kadut to move furniture around the facility [4]. By utilizing IIoT, they are able to increase their operational efficiency by 50%. This could allow Singapore companies such as Star Living to upscale and expand their operations to foreign markets as well. As such, IIoT is an essential part for the growth of Singapore as a nation.

Multi-Access Edge Computing (MEC) is a concept introduced by the European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG) as a catalyst for the implementation of IoT and IIoT systems [6]. The fundamental concept of MEC is to implement cloud server computing capabilities in edge servers which are closer to the end users [7]. By implementing MEC in the network, it can reduce network congestion and minimize latency. Caching a certain amount of content in edge servers minimizes the need for users to connect to the cloud server for the required content. Furthermore, transmitting to edge servers instead of the cloud server reduces the time between request and receive of the content, thus reducing latency.

However, implementing MEC consumes a large amount of energy. Edge caching is the process in which popular content and computations are stored in the cache located in the edge servers. Due to limited cache space in the edge servers, only a limited amount of content such as data and computations can be stored in the cache. Coupled with the continuously changing popularity of content with time and preference, the cache would be required to be periodically updated to adapt to the changing preferences. Constant updates of the cache would result in more energy consumed by the system [8]. Furthermore, energy is also consumed continuously within IIoT systems to sustain the transmissions within the system, as the IIoT devices may need to often obtain necessary content from the BS. Thus, it is necessary to maximize EE, which can be defined as the ratio of the average achievable rate to the total power consumed by the system. The objective of maximizing EE is to minimize the energy consumption cost while maintaining a stable and acceptable transmission rate within the network. By minimizing the energy consumption cost of the system, more organizations will be willing to utilize MEC-enabled IIoT networks as a method to improve the productivity and efficiency of their activities. While organizations already utilizing MEC-enabled IIoT networks can save on costs, that they can use the extra funds for other activities or projects. Overall, reducing energy consumption in MEC-enabled IIoT networks benefits organizations, whether if it’s an organization which only started using MEC-enabled IIoT networks or organization which has already implemented its own one.

1.1. Literature Review

EE can be defined as the ratio of the average achievable transmission rate in the system to the power consumed by the system. EE has been an issue when it comes to IIoT due to the large amount of energy consumed to maintain it. Since devices in IIoT systems are constrained by computation and communication resources, energy consumption can limit the lifetime of the IIoT system [9].

Furthermore, devices in IIoT with the capabilities of sensing, communication, processing or computation also consumes a lot of energy which eventually increases the carbon footprint of deploying these IIoT systems [10]. This will be difficult especially for corporates due to carbon taxes. In 2024, carbon taxes in Singapore has increased to $25 per tonne of emissions from $5 per tonne previously, which is a 400% increase in cost per tonne [11]. This further incentivizes the need for optimization of EE to mitigate tax burdens and promote sustainable practices within organizations.

Various approaches exist to optimize EE in IIoT systems. For example, works such as [12] discuss about implementing intelligent edge computing in IIoT system capable of delegating AI tasks from the cloud server to the edge nodes using an online scheduling strategy which is stated to consume 20% less energy compared to the static scheduling strategy. Another work shares regarding a Heuristic and Opportunistic Link Selection Algorithm (HOLA) to optimize EE through the opportunistic transfer of IoT device data to smart devices carried by the employees in a factory environment [13]. In addition, these smart devices, equipped with various radio links like Bluetooth, Wi-Fi, and 3G/4G LTE, utilize heuristic methods to determine the most optimal link for transmitting content to the Cloud, taking into consideration both link quality and energy consumption.

Recent studies explore integrating the Digital Twin technology, which is the digital virtualization of physical objects, along with the Raindrop Algorithm, for offloading optimization [14]. By optimizing the offloading process of the tasks in the IIoT system, the system would be able to minimize the task completion time, optimize the resource distribution and minimize the energy consumption.

1.2. Objective

The primary objective of this project is to propose a framework for optimizing instantaneous EE of edge caching in the 5G-and-beyond IIoT Networks. Given the growing significance of IIoT in industrial processes, optimizing EE in IIoT networks is crucial for reducing energy costs, mitigating carbon emissions, and enhancing operational efficiency.

This report aims to achieve these objectives:

1. **Establish Benchmark Values:** The first objective is to obtain benchmark values representing the average optimal EE in IIoT networks. These benchmarks serve as a reference point for evaluating the performance and efficiency of the proposed framework. By benchmarking against established standards, we can assess the effectiveness of our proposed framework.
2. **Machine Learning-Based Optimization:** The second objective involves training different machine learning models to directly optimize EE. These models predict optimization variables such as transmit power, SBS cache size, and MBS cache size. By leveraging environmental variables such as estimated channel gains, estimation errors, and distances between Automated Mobile Robots (AMRs) and other network components, our models—specifically Deep Learning (DL) and TabNet—aim to enhance EE through intelligent decision-making.
3. **Performance Comparison:** Lastly, we will compare the performance and efficiency of our machine learning models with the brute force method used to obtain benchmark values. This comparison will demonstrate the efficacy of our proposed framework in achieving optimal EE in IIoT networks. By evaluating the computational efficiency and accuracy of our models relative to traditional methods,

we can highlight the practical benefits of adopting machine learning-based approaches in IIoT optimization.

2. System Model

The system model is shown in Figure 2 below. The system model involves four key entities. These entities include the typical AMR, SBS, MBS and IAMRs. Both SBS and MBS has a MEC server and are able to cache content. In this simulation, there will be two IAMRs, one SBS and one MBS. The main focus in this system model would be the typical AMR. There will be variables which will be fixed in the simulation. These variables include the maximum allocated power to the system, the total number of files in the cloud server and the SBS and MBS cache sizes. The SBS and MBS are connected to the cloud server through wired backhaul, where there are physical wires connecting the BS to the cloud server. The IAMRs are communicating in the same time-block frequency as the typical AMR, which could cause interference and noise in the transmission between the typical AMR and the SBS and MBS in our system model.

During downlink transmission between the typical AMR and the BS, they follow a certain association strategy. The association strategy dictates how the typical AMR acts when searching for desired content from the BS or the cloud server. These association strategies include the Distance-Based Association Strategy, CBAS and Cloud-Based Association Strategy.

This setup uses CBAS as the strategy. CBAS follows the rule where the typical AMR will search through the first BS before continuing to the next BS if the file is not found in the first BS. In this scenario, the typical AMR would search through the SBS before continuing to the MBS and then connect to the Cloud Server through the MBS.

The assumption in the system model is that the typical AMR is stationed within an indoor industrial setting. The typical AMR requires the multimedia content for the monitoring and quick response to any occurrence of predictable and unpredictable scenarios which could potentially occur within the industrial grounds, whether it is the ground, machinery or activities occurring within the industrial building [15].

EE of the transmissions between the typical AMR and the other players in the system model is the main focus of this work. Nakagami-m fading, along with imperfect channel state information (CSI), are used as the small-scale fading in the indoor industrial setting, which could impact the energy expenditure within the system model [16]. Line-of-Sight blockages by metallic surfaces are also considered in the system model, where it is shown by the light red walls in Figure 2.

The popularity of the files in the cloud server follows a Zipf distribution, similar to [17]. Zipf distribution follows Zipf's Law which states that the frequency in which a content is requested for is inversely proportional to its long-term popularity [18]. Furthermore, the files are also cached in the SBS and MBS based on the Most Popular Content (MPC) strategy, where only the most popular contents are cached in the edge servers in the BS [19].

The delay for the wired backhaul connection between the cloud server and the MBS is generated using the gamma distribution model, similar to works such as [20].

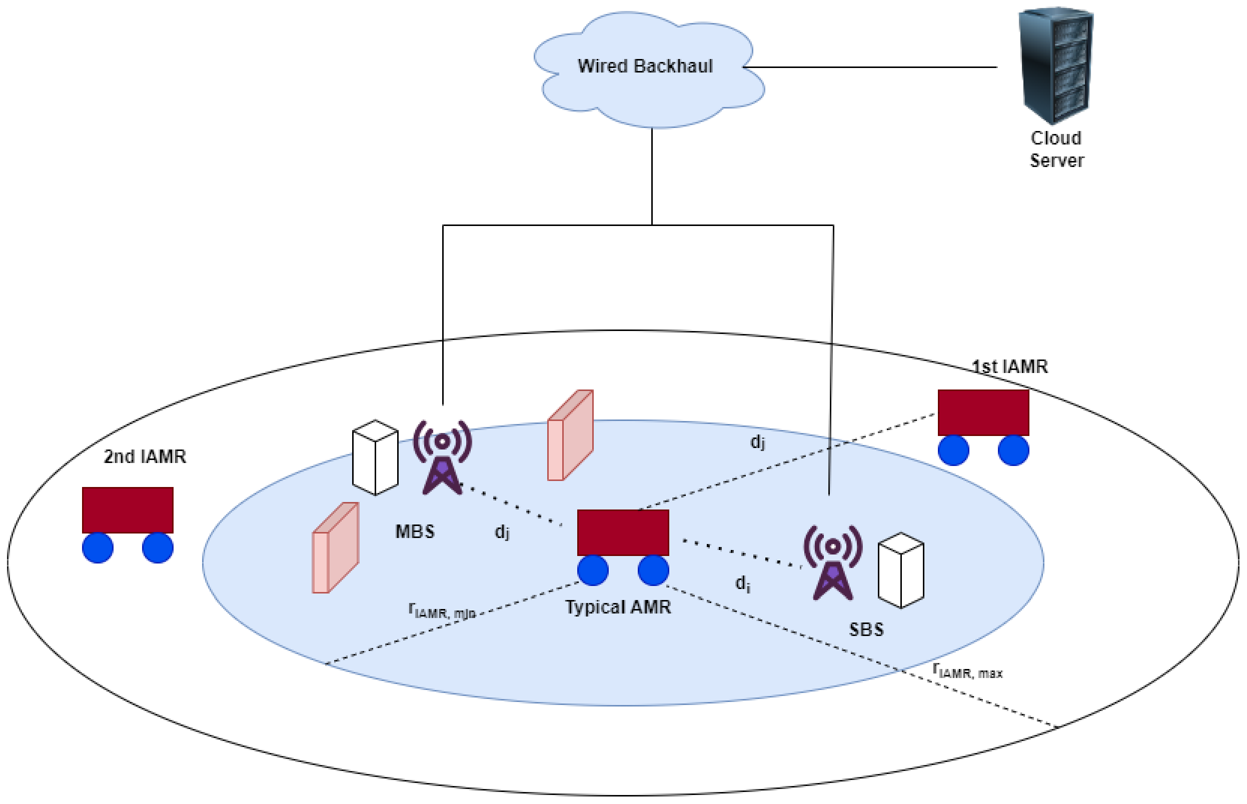


Figure 2: System Model

Notations	Descriptions
d_i	Distance between the typical AMR to the SBS
d_j	Distance between the typical AMR to the MBS
d_z	Distance between the typical AMR to the IAMR
$r_{IAMR,min}$	Minimum radius where the IAMR is located
$r_{IAMR,max}$	Maximum radius where the IAMR is located

Table 1: List of Notations for System Model

2.1 Optimization Problem

The optimization problem of this project is given as:

$$\max_{P_t{}^*, K_{SBS}{}^*, K_{MBS}{}^*} EE_{CBAS}$$

$$\text{s.t.:}$$

$$0 \leq P_t \leq P_{allocated}$$

$$K_{SBS}\,,\ K_{MBS}\ \geq 0$$

$$K_{SBS} + K_{MBS}\ \leq\ K_{total}$$

Where EE_{CBAS} is the EE of the transmission based on CBAS. $P_{allocated}$ is the allocated power to the system, in which the transmit power cannot exceed the allocated power to the system. K_{total} is the total number of files found in the Cloud Server. We will be looking at optimizing instantaneous EE by optimizing the parameters shown in the optimization problem, which are $P_t, K_{SBS}\ and\ K_{MBS}$. The relationship between the maximum transmit power (P_{max}), the allocated transmit power ($P_{allocated}$), and the transmit power (P_t)is shown in the equation below.

$$0 \leq\ P_{allocated} \leq P_{max}$$

$$0 \leq\ P_t \leq\ P_{allocated}$$

3. Methodology

3.1 Benchmark Initialization

A python function is utilized to generate the necessary environmental variables within the simulation between the typical AMR and the other players in the system model, such as the estimated channel gains, estimation errors and distances. These variables were used to compute the SINR, which in turn were used to calculate the instantaneous EE. A dataset containing the values of instantaneous EEs for different combinations of transmit power, SBS cache size and MBS cache size was generated. This dataset was generated through the brute force method, where it exhaustively searched through every possible combination of the three previously stated combinations and obtain the respective instantaneous EE values as well as average EE values, which is the average of the optimal instantaneous EE values over a number of samples. The average EE values will be used as a measure of the performance of the machine learning models.

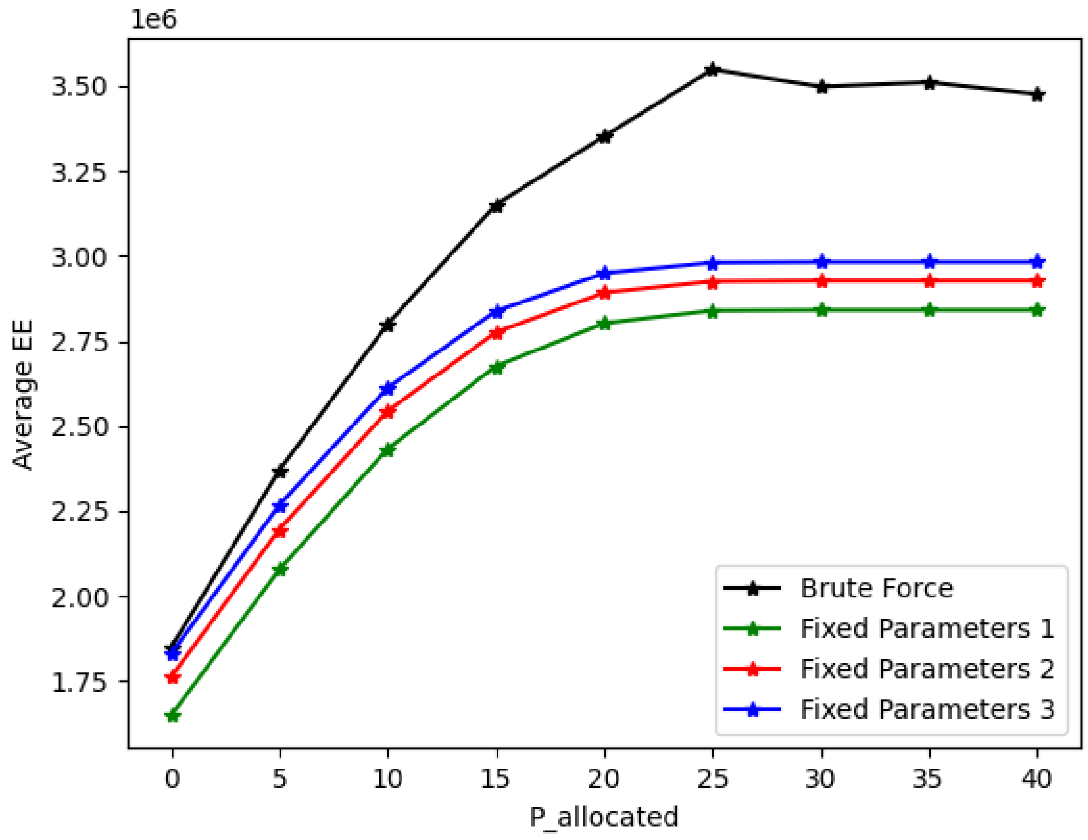


Figure 3: Brute Force Method vs Fixed Parameters

Figure 3 shows the difference between utilizing the brute force method to obtain the average EE values as compared to using fixed parameters method, where the values SBS cache size (K_{SBS}) and MBS cache size (K_{MBS}) were fixed while only allowing the transmit power (P_t) to vary, as compared to the brute force method where K_{SBS}, K_{MBS} and P_t varies. Brute force method was able to output higher average EE values compared to the fixed parameters method. The parameters given to the brute force method and the fixed parameters are given in Table 2.

Method	K_{SBS}	K_{MBS}
Brute Force	$0 \leq K_{SBS} \leq 10$	$0 \leq K_{MBS} \leq 200$
Fixed Parameters 1	10	200

Fixed Parameters 2	15	250
Fixed Parameters 3	20	300

Table 2: List of Brute Force vs Fixed Parameters values

The disparity between the brute force method and the fixed parameters method for the allocated power of 40 dBm as shown in Figure 3 is mainly due a much more significant increase in the energy consumption by the system for the fixed parameter method compared to the brute force method. Since the K_{SBS} and the K_{MBS} values are fixed, the power consumed by the caches is not adjusted to the minimal possible consumption, as compared to the brute force method, in which the values are adjusted to obtain the minimum power consumed by the caches. Hence, in the scenario in which the average achievable rate is similar for both fixed parameter method and brute force method, the energy efficiency would be higher for brute force method, as it iterates through the different values of K_{SBS} and K_{MBS} values to obtain the minimum total energy consumption, compared to the fixed parameter method, especially considering the direct proportionality between the allocated transmit power and the total energy consumed by the system.

3.2 Training Dataset Generation

A function, where an instantaneous SINR calculator function is utilized, was created with the purpose of computing the instantaneous EE between the typical AMR and the other players in the simulation. This function was used to generate a dataset of optimal instantaneous EE values given the different combinations of transmit powers, SBS cache sizes and MBS cache sizes, as well as the environmental variables in the system model which impacts the calculation of instantaneous EE. The pseudocode for data generation is given in Algorithm 1.

Algorithm 1: Training Data Generation

```
1.  for each $P_{allocated}$ = 0…$P_{max}$, $P_{allocated}$ = $P_{allocated}$ + step size
2.      Initialize environmental variables
3.      Initialize gamma distribution of delays
4.      Initialize zipf distribution
5.      Initialize file requests
6.      for each N = 1…$N_{sample}$:
7.          for each $K_{SBS}$ = 0…$K_{SBS,max}$, $K_{SBS}$ = $K_{SBS}$ + step size:
8.              for each $K_{MBS}$ = 0…$K_{MBS,max}$, $K_{MBS}$ = $K_{MBS}$ + step size:
9.                  for each $P_t$ = 0…$P_{allocated}$, $P_t$ = $P_t$ + step size:
10.                     COMPUTE instantaneous $EE_{CBAS}$
11.         COMPUTE optimal instantaneous $EE_{CBAS}$
12.         COMPUTE optimal average $EE_{CBAS}$
13. Write $P_{allocate}$, $P_t$, $K_{SBS}$, $K_{MBS}$, optimal instantaneous $EE_{CBAS}$, environmental
    variables into CSV file
14. Write $P_{allocate}$, optimal average $EE_{CBAS}$ in a different CSV file
```

Algorithm 1: Pseudocode for Training and Testing Dataset Generation

These values were recorded in a CSV file along with the optimal instantaneous EE values. Two files will be generated through this process, one CSV file containing the environmental variables, $P_t, K_{SBS}, K_{MBS}, P_{allocated}$ and instantaneous EE, while the other CSV files contains the $P_{allocated}$ and its respective average optimal instantaneous EE values for a number of samples.

The training dataset generated was then used to train the models to predict the optimal parameters which affects the instantaneous EE based on the optimization problem, which are P_t, K_{SBS}, and K_{MBS} given the environmental variables within the system model.

The total time taken for the brute force method to generate the two testing datasets used in this project are 715.23 seconds and 1084.56 seconds respectively.

The training dataset was split into train and validation sets with the percentages of 80% and 20% respectively. The inputs to the machine learning models consist of 16 columns while the outputs consist of 3 columns which are P_t, K_{SBS}, and K_{MBS}.

3.3 Machine Learning Models and Training

Before feeding the input into the training of the machine learning models, preprocessing was done to the input data. Columns which involve the relationship between the typical AMR and the IAMRs, such as the channel gain and estimation error, are split into two different values, since there are two IAMRs in the system model. Furthermore, initially, the "Hit Status" column in the input data is categorical in nature, which is either "SBS", "MBS" or "Cloud Server", one-hot encoding is executed on this input [21]. Since the DL Algorithm is not able to process nominal data such as the "Hit Status" column, it is necessary to convert the content of this column into numerical values for the DL algorithms to process. The "Hit Status" column is split into three different columns, which are "Hit Status SBS", "Hit Status MBS" or "Hit Status Cloud". If the file was found in the SBS, then "Hit Status SBS" will have a value of 1 while the others will have a value of 0. This applies to the other scenarios as well. The input data is then normalized using the standard scaler from sklearn while the output data is normalized by dividing P_t by $P_{allocated}$ and dividing K_{SBS} and K_{MBS} by their maximum value.

Supervised learning was implemented for the machine learning models. Supervised learning is a machine learning method where both the input features and targeted outputs are known to the model. The training aims to minimize the loss or difference between the predicted output generated by the model and the actual output given the input features. The training was implemented with an EarlyStopper with patience of 10. Implementing an EarlyStopper with patience of 10 ensured that the models stopped training after there was no decrease in the validation loss observed after 10 epochs from the lowest validation loss previously recorded. This is to ensure that overfitting does not occur during the training of the models. Overfitting is defined as the phenomenon where the model tends to memorize the trends in the training data instead of learning from them instead [22]. This causes the model to perform well on the training data but performs poorly on unseen data. The model saved during the training of the machine learning models would be the model generated on the epoch in which the average validation loss, which is the difference between the predicted output of the model and the actual output, is at the minimum point.

4. DL Model

One of the models we will be exploring to solve this problem is the Multi-Layer Perceptron DL Model. There are a few reasons why DL Model was chosen for this task.

Firstly, DL models offer scalability and flexibility, allowing them to accommodate varying degrees of complexity in IIoT systems. This scalability enables the DL models to adapt to different deployment scenarios and network configurations, ensuring robust performance across diverse scenarios.

Furthermore, the simplicity of the DL models contributes to their appeal for EE optimization in IIoT networks. The straightforward architecture of the DL model facilitates rapid deployment and implementation, requiring minimal computational resources for training and inference. This efficiency translates into faster model deployment and real-time decision-making, critical for optimizing EE in dynamic IIoT environments.

The DL model was trained to predict the P_t, K_{SBS} and K_{MBS} values given the environmental variables in the system model as the input. Each hidden layer is followed up by a ReLU activation function before continuing to the next hidden layer or to the output layer. For the output layer, clamping from the torch library is used on the P_t output such that $0 \leq P_t \leq P_{allocated}$. While ReLU activation function was implemented on the K_{SBS} and K_{MBS} outputs such that the outputs will always be equal to or greater than zero. The pseudocode for the DL model is given in Algorithm 2.

```
Algorithm 2: DL Model
---------------------------------------------------------------
Inputs:
        n_list = list of neurons inside the hidden layers
```

```
    input_size = dimension of input data
    output_size = dimension of output data
1.  Layers = []
2.  Layers ← nn.Linear(input_size, n_list[0])                    //initialize input layer
3.  Layers ← ReLU layer
4.  **for each** i = 1…len(n_list):
5.          Layers ← nn.Linear(n_list[i – 1], n_list[i])         //initialize hidden layers
6.          Layers ← ReLU layer
7.  **forward method**
8.          min = 0
9.          max = 1
10.         **torch.clamp** $P_t, K_{SBS}$ and $K_{MBS}$, range: min…max     //limit output to 0-1
11.         output = **torch.cat**($P_t, K_{SBS}$ and $K_{MBS}$)  //concatenate the outputs together
```

Algorithm 2: Deep Learning Algorithm

5. TabNet

Another machine learning model we will be looking into for this project is TabNet. TabNet is a deep tabular data learning architecture. Unlike normal DL models, TabNet is designed to work better on structured data represented in tables.

TabNet uses sequential attention mechanism to select out the most prominent features for reasoning at each decision step. This method enables interpretability and improved learning since the model focuses more on the most impactful features in the input dataset [23]. In the context of our project, where we are dealing with 16 different input features for model training, TabNet's attention mechanism enables it to prioritize and leverage the most influential variables, leading to more accurate predictions and insights into EE optimization.

Moreover, the interpretability offered by TabNet's attention mechanism is particularly beneficial for understanding the underlying factors influencing the instantaneous EE computation. By identifying and highlighting the key features driving energy consumption patterns, TabNet enables stakeholders to gain valuable insights into optimization strategies and decision-making processes.

For this project, we will be using a simplified version of the TabNet model architecture. Instead of multiple decision steps in the model, this simplified version will be just going through one cycle of decision step before predicting the output of the model. Our TabNet model consists of the initial decision layer, feature transformation layer and the prediction layer. The pseudocode for the TabNet model is shown in Algorithm 3.

Algorithm 3: TabNet Model

```
Inputs:
        input_dim = dimension of input data
```

```
    output_fim = dimension of output data
1.  **initialize** input_dim and output_dim
2.  **initialize** initial decision layer (linear layer and 1D batch normalization layer)
3.  **initialize** feature transformation layer (linear layer and 1D batch normalization
    layer)
4.  **initialize** prediction layer of $P_t, K_{SBS}$ $and$ $K_{MBS}$
5.  **forward method**
6.          decision = initial decision layer (input)
7.          mask = softmax (decision)
8.          masked_input = input * mask              //Applying mask to input
9.          features = feature transformation layer (masked_input)
10.         $P_t, K_{SBS}$ $and$ $K_{MBS}$ = prediction layer (features)
11.         **torch.clamp(**$P_t, K_{SBS}$ $and$ $K_{MBS}$**),** min = 0, max = 1 //Clamping the
    output of the model
```

Algorithm 3: TabNet Model Structure

6. Results

6.1 DL Model

P_{max} is set to be at 40 dBm. $P_{allocated}$ varies between 0 and P_{max} with a step size of 5. The total number of files in the cloud server is set to be 1000 while the cache sizes in the SBS and MBS are set to be 10 and 200 respectively. The equations below are the variation of data used for the training dataset and the testing dataset.

$$0 \le P_t \le P_{allocated}, step\ size = 5$$

$$0 \le K_{SBS} \le 10, step\ size = 2$$

$$0 \le K_{MBS} \le 200, step\ size = 20$$

The training dataset consists of 10,000 data rows per $P_{allocated}$ value, while the two testing datasets consist of 4000 and 6000 data rows per $P_{allocated}$ value respectively.

The different columns of input data fed to train the models are given in Table 3 below.

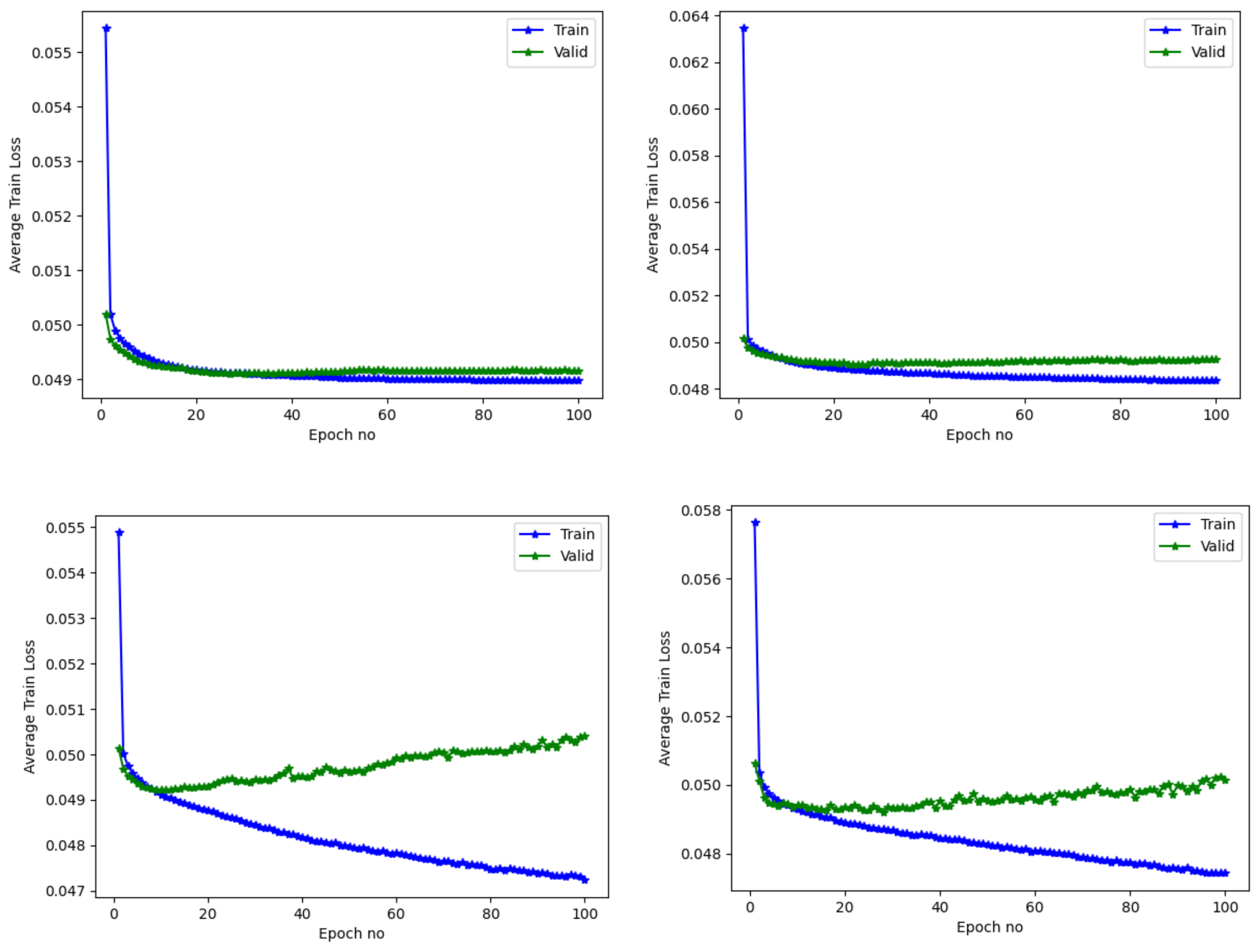
Variables	Description
Channel Gain i	Estimated channel gain between the typical AMR and the SBS
Channel Gain j	Estimated channel gain between the typical AMR and the MBS
Channel Gain z	Estimated channel gain between the typical AMR and the IAMRs, split into two separate columns
Estimation Error i	Estimation error between the typical AMR and the SBS
Estimation Error j	Estimation error between the typical AMR and the MBS
Distance from SBS	Distance between the typical AMR and the SBS
Distance from MBS	Distance between the typical AMR and the MBS

Distance from zth IAMR	Distance between the typical AMR and the IAMRs, split into two separate columns
Beta i	Line-of-sight (LOS) occurrence between transmission of typical AMR and SBS
Beta j	LOS occurrence between transmission of typical AMR and MBS
T_bs	Delay for wired backhaul link between the BS and the cloud server
Hit status (SBS, MBS or Cloud Server)	Whether the typical AMR finds the content in the, split into three separate columns according to the hit record

Table 3: List of Input Variables for training

Figure 4 shows the training graphs of the different number of hidden layers of the DL model.

The models were trained for 100 epochs to ensure that the global minimum has been reached.



<u>**Figure 4**: Training Epochs Graph for 1 Hidden Layer (Top Left), 2 Hidden Layers (Top Right), 4 Hidden Layers (Bottom Left), 10 Hidden Layers (Bottom Right) Perceptron</u>

Firstly, we will be looking at the results of the predictions made by the DL model. Figure 5 shows the average EE values generated from the $P_t, K_{SBS}\ and\ K_{MBS}$ predictions made by the model on the 4000 sample testing data.

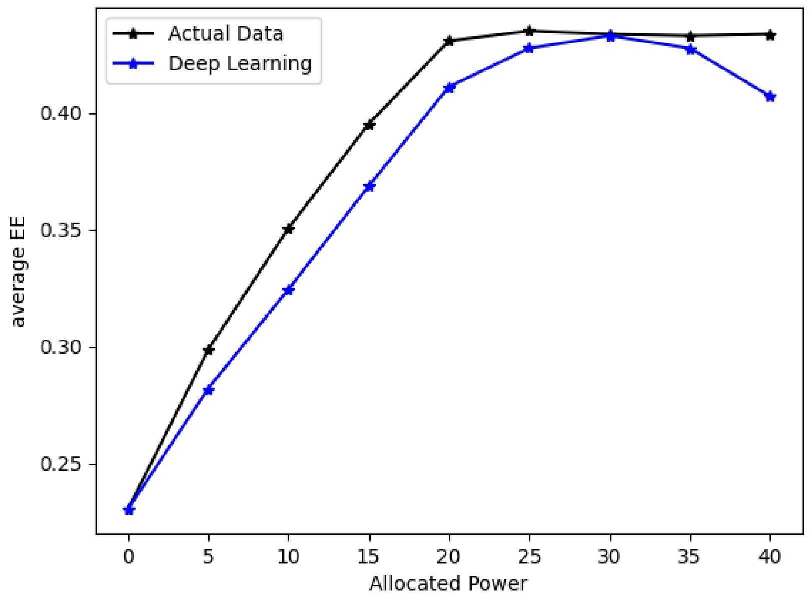


Figure 5: Average EE performance of Deep Learning Model

From Figure 5, it is observed that the DL model performs less accurately compared to the brute force method, which is shown by the black line plot. However, the DL model was able to learn the trend of the average EE properly as observed from the graph above. The trend learnt is that as the allocated power, $P_{allocated}$, increases, there is also an increase in the average EE values. The time taken for the model to output the optimal values of $P_t, K_{SBS}\ and\ K_{MBS}$ is 0.908 seconds. The absolute average difference of the average EE values between the brute force method and the DL model is 0.014333 MB/J. The average percentage difference between the brute force method and the DL model is 3.728%. The 30 dBm allocated power mark has the lowest difference in average EE between the brute force method and the DL model.

Figure 6 shows the performance of the DL model compared to the brute force method for the

6000 samples testing set.

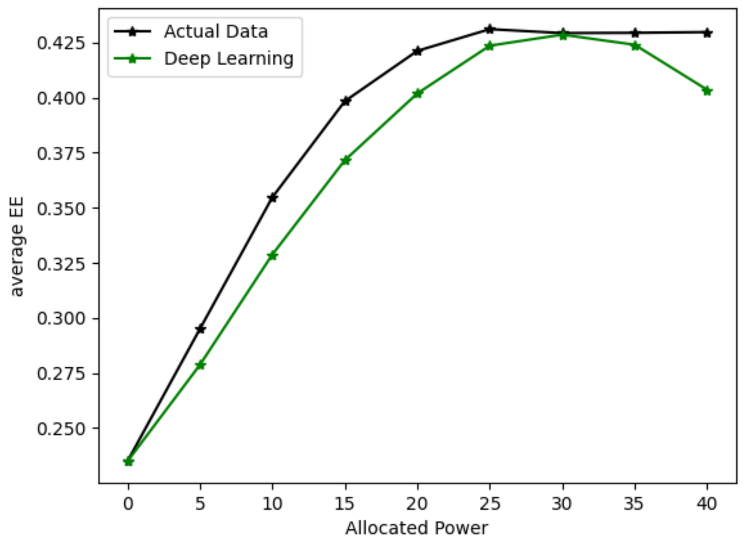


Figure 6: 6000 Sample Set for DL Model

From Figure 6, it can be observed that the DL model is still able to follow the trend of the brute force method. For the 6000 samples set, the absolute average difference is 0.014251 MB/J while the average percentage difference is 3.716%. The time taken for the DL models to output the optimal parameters for the 6000 samples set is 1.338 seconds.

Furthermore, to see if the DL model can also be used for other scenarios other than varying $P_{allocated}$, the DL model was also tested using varying allocated SBS cache sizes and MBS cache sizes. The K_{SBS} or K_{MBS} are varied according to the $K_{SBS,allocated}$ or $K_{MBS,allocated}$, while the $K_{SBS,allocated}$ and $K_{MBS,allocated}$ also varies based on a certain range. The P_t only varies from the value of 0 to P_{max} with a step size, instead of varying according to the $P_{allocated}$ value. Only the allocated SBS cache size or the allocated MBS cache size can vary at one point of time, and not both at the same time. Figure 7 shows the performance of the DL model against the brute force method for varying $K_{SBS,allocated}$ and varying $K_{MBS,allocated}$.

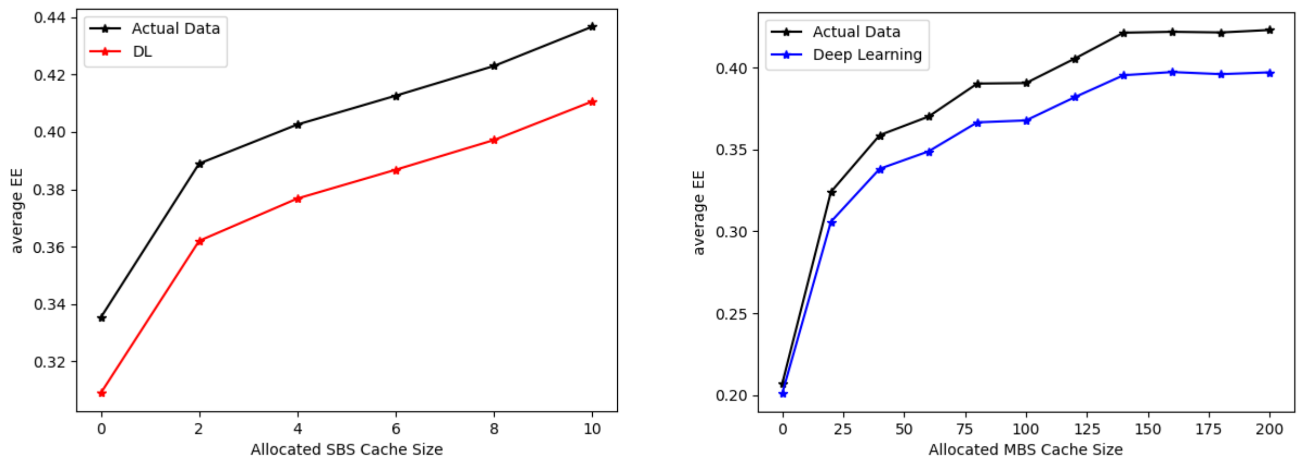


Figure 7: Allocated SBS Cache Size (Left) and Allocated MBS Cache Size (Right) Scenario for DL model

As shown in Figure 7, the DL model was also able to successfully learn the trends from the brute force method. The $K_{SBS,allocated}$ graph has an absolute average difference of 0.02611 MB/J and an average percentage difference of 6.581%. The $K_{MBS,allocated}$ graph has an absolute average difference of 0.02165 MB/J and an average percentage difference of 5.626%.

5.2 TabNet Model

Next, we will be looking at the TabNet model performance.

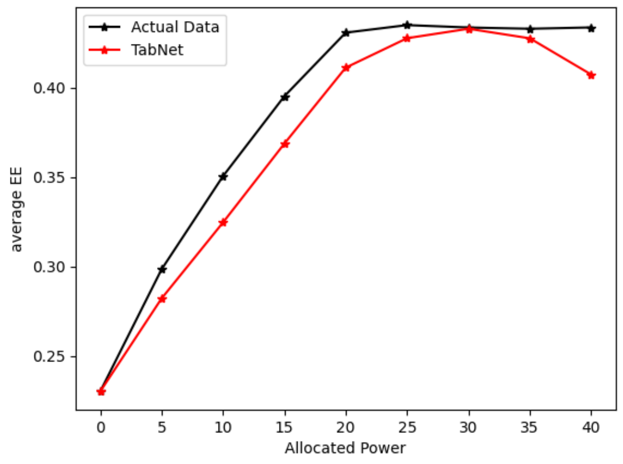


Figure 8: TabNet Model Performance

Figure 8 shows the performance of the TabNet model on the 4000 samples dataset. It is shown that TabNet was also able to properly learn the trend of the data obtained from the brute force method. The absolute average difference between the TabNet model and the brute force method is 0.01421 MB/J while the percentage difference is shown to be 3.695%. The TabNet model was able to complete the dataset in 1.145 seconds.

Figure 9 shows the performance of the TabNet model on the 6000 samples dataset.

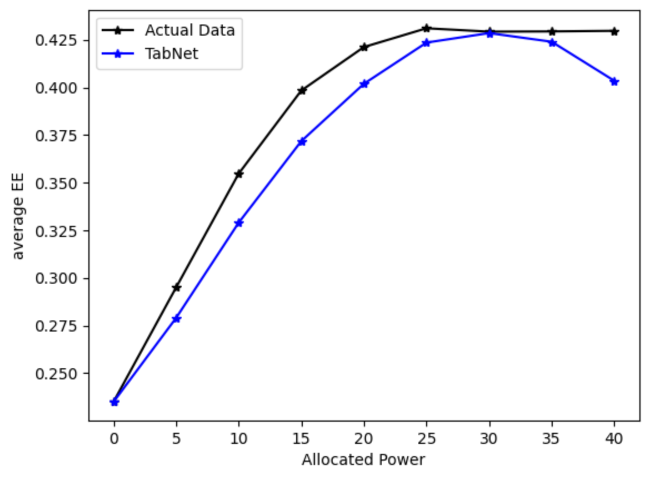


Figure 9: 6000 Sample Set for TabNet Model

The average difference between the TabNet model and the brute force method is 0.01417 MB/J for the 6000 samples dataset. The percentage difference between the TabNet model and the brute force method is 3.691% for the 6000 samples dataset. The time taken for the TabNet model to complete predicting the P_t, K_{SBS} and K_{MBS} parameters for the 6000 samples dataset is 1.439 seconds.

Additionally, Figure 10 shows the performance of TabNet based on $K_{SBS,allocated}$ and $K_{MBS,allocated}$ datasets.

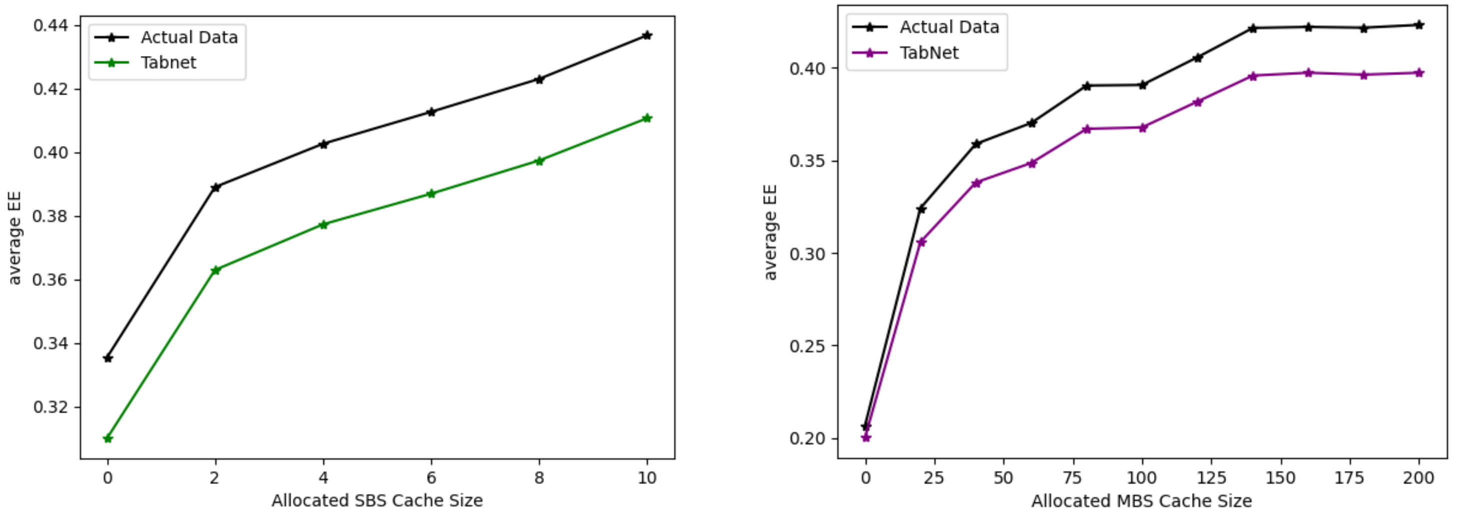


Figure 10: Allocated SBS Cache Size (Left) and Allocated MBS Cache Size (Right) Scenario for TabNet model

As shown in Figure 10, TabNet was also able to trace on the trend shown by the brute force method. The $K_{SBS,allocated}$ graph has an absolute average difference of 0.02571 MB/J and an average percentage difference of 6.473%. The $K_{MBS,allocated}$ graph has an absolute average difference of 0.02174 MB/J and an average percentage difference of 5.660%.

7. Analysis

Table 4, Table 5 and Table 6 summarize the findings from the experimentation.

Method	**Time taken to complete 4000 samples per $P_{allocated}$ value (seconds)**	**Time taken to complete 6000 samples per $P_{allocated}$ value (seconds)**
Brute Force	715.23	1084.56
DL	0.908	1.338
TabNet	1.145	1.439

Table 4: Generation Method and Time Taken to Complete

Method	**Percentage difference in value to brute force method for 4000 sample dataset**	**Percentage difference in value to brute force method for 6000 sample dataset**
DL	3.728%	3.716%
TabNet	3.695%	3.691%

Table 5: Percentage Difference on Allocated Transmit Power data

Method	**Percentage difference in value to brute force method for $K_{SBS,allocated}$**	**Percentage difference in value to brute force method for $K_{MBS,allocated}$**
DL	6.581%	5.626%
TabNet	6.473%	5.660%

Table 6: Percentage Difference on allocated SBS Cache Size and MBS Cache Size data

As observed Table 4, the DL model and the TabNet model performs much faster compared to the brute force method. The DL model and TabNet model did not perform as well as the brute force method, however it is only 3.6% to 3.8% difference in accuracy as shown in Figure 5, which is negligible as compared to the difference in time taken. Thus, the DL model and TabNet model proved to be a better approach compared to brute force method. As shown in Table 6, there is also only a small difference of 5-7% accuracy on the $K_{SBS,allocated}$ and $K_{MBS,allocated}$ data, which shows that the models can also be used on other situations or types of data.

Comparing between the DL model and the TabNet model, the DL model is slightly faster in completing the task as compared to the TabNet model, but less accurate in its output. With the current differences, there is no model that stands above the other. However, one model could be preferred than the other depending on the situation. For example, if the requirement of the model is on speed, then the DL model would have an edge compared to the TabNet model. Not only that the DL model was faster in generating the outputs, it is also simpler to code and implement compared to the TabNet model, which means that it would be faster to deploy the DL model compared to the TabNet model. On the flip side of the coin, if the goal of the model is to get the output parameters as optimal as possible, then the TabNet model would be the preferred option compared to the DL model.

Furthermore, the TabNet model also has another advantage on the DL model in terms of the TabNet model having an attribute called interpretability. In TabNet models, feature selection is part of the process in which features are weighed based on their impact on the desired output of the model through the attention mechanism in TabNet models [21]. This process can give meaningful insights to users regarding the relationships between the features and the outputs which could help users understand the logic behind the model's predictions and to improve the model further.

Another observation during the experimentation is that despite the machine learning architecture used, whether it is DL or TabNet, the average optimal EE values predicted by these models are closest to the brute force method output, or the ground truth, when the $P_{allocated}$ value is equal to 30 dBm. From this observation, it can be deduced that 30 dBm is the optimal operating point for allocated power to the MEC-enabled IIoT system in terms of optimal instantaneous EE, where it is at this point that there is the best balance between the average achievable rate in the system and the energy consumption of the system.

8. Conclusion

In this project, machine learning models such as DL model and TabNet model have been introduced as frameworks for optimizing instantaneous EE for edge caching in 5G-and-beyond IIoT networks. We explored using brute force method to obtain the benchmark as well as training data and testing data. Preparing the dataset before training of the models has also been discussed in this paper. A summary of the comparison between the machine learning models and the brute force method can be found in Table 4 and Table 5.

The results of the experiment have shown that the machine learning models proposed, which are the DL model and TabNet model, were better methods to use compared to the traditional brute force method for predicting parameters involved in the optimization problem. The machine learning models were able to complete predicting the optimal outputs given the inputs in significantly less time compared to brute force method given the same number of samples. This could be done while only losing around 3-4% of accuracy in the output compared to the brute force method output, which was considered as the ground truth during the experimentation.

Furthermore, regarding on whether to use DL model or TabNet model, it depends on the situation in which it would be implemented in. If there is more focus on speed in deployment and execution, DL would be the recommended model to use. However, if there is more focus on accuracy and analysis of relationship between inputs and outputs, TabNet model would be the recommended model to deploy.

Overall, the proposed machine learning models were able to satisfy the requirements of the project, which is to improve on the brute force method on optimizing EE in 5G-and-beyond IIoT networks. By enabling faster and more accurate optimization of EE, these models enhance operational efficiency and reduce energy consumption, thereby lowering energy costs associated with IIoT deployment in industrial settings. Moreover, their scalability and adaptability make them well-suited for accommodating future growth in IIoT deployments while minimizing environmental impact.

Nevertheless, there are still more which could be done regarding the machine learning models to further improve its utility in optimizing the instantaneous EE in the IIoT system. For example, the TabNet model that was used in this project is a simplified version with only one cycle of decision step. With a more complex architecture and more decision steps, the TabNet model could potentially learn more about the relationship between the features and the outputs and better classify the features which has the largest impact on the output of the model, improving the overall performance of the TabNet model. A more complex DL model can also be implemented specifically for this task, with hyperparameter tuning, to obtain better results on the predictions of the model. These models could also be extended outside of only optimizing EE but also components within the IIoT network such as latency or resource utilization within the IIoT network such as computing, storage, and communication resources.

In conclusion, the utilization of machine learning models for EE optimization in IIoT networks offers promising benefits for industrial deployment. By leveraging these models, organizations can achieve optimal EE instances more efficiently, thereby minimizing energy waste and reducing deployment costs. As we continue to refine and expand the capabilities of machine learning models in IIoT deployment, we unlock new opportunities for further enhancement in operational efficiency and driving innovation in industrial processes.

Appendix

Model	Time Taken
1 Hidden Layer	0.561s
2 Hidden Layer	0.503s
4 Hidden Layer	0.487s
10 Hidden Layer	0.539s

Table 7: Deep Learning Different Number of Hidden Layers Time Taken

Model	Average Difference
1 Hidden Layer	0.01433
2 Hidden Layer	0.01457
4 Hidden Layer	0.01463
10 Hidden Layer	0.01461

Table 8: Deep Learning Different Number of Hidden Layers Absolute Average Difference

References

1. T. Qiu, J. Chi, X. Zhou, M. Atiquzzaman, and D. O. Wu, “Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges”, IEEE Comm. Surv. Tutor., vol. 22, no. 4, pp. 2462-2488, 4th Quart., 2020.

2. Industrial IoT - Singapore, , n.d.. [Online]. Available: https://www-statista-com.remotexs.ntu.edu.sg/outlook/tmo/internet-of-things/industrial-iot/singapore

3. "Smart Nation Strategy - November 2018," Smart Nation Singapore, Nov. 2018. [Online]. Available: https://www.smartnation.gov.sg/files/publications/smart-nation-strategy-nov2018.pdf.

4. Rosalind Ang, “More tech, less physical labour at Star Living’s new furniture warehouse”, The Straits Times, Oct 25, 2023. [Online]. Available: https://www.straitstimes.com/business/more-tech-less-physical-labour-at-star-living-s-new-furniture-warehouse [Accessed: March 21, 2024]

5. X. Hou, Z. Ren, K. Yang, H. Zhang, and Y. Xiao, "IIoT-MEC: A Novel Mobile Edge Computing Framework for 5G-enabled IIoT," in Proceedings of IEEE Wireless Communications and Networking Conference (WCNC), 2019, pp. 1-7. DOI: 10.1109/WCNC.2019.8885703.

6. P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttilla, and T. Taleb, “Survey on Multi-Access Edge Computing for Internet of Things Realization”, IEEE Comm. Surv. Tutor., vol. 20, no. 4, pp. 2961-2991, 4th Quart., 2018.

7. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges”, IEEE Internet Things J., vol. 3, no. 5, October 2016.

8. M. Yan, C. A. Chan, W. Li, L. Lei, A. F. Gygax, and C.-L. I, “Assessing the energy consumption of proactive mobile edge caching in wireless networks,” IEEE Access, vol. 7, pp. 104394–104404, 2019.

9. W. Mao, Z. Zhao, Z. Chang, G. Min, and W. Gao, “Energy-Efficient Industrial Internet of Things: Overview and Open Issues,” IEEE transactions on industrial informatics., vol. 17, no. 11, pp. 7225–7237, 2021, doi: 10.1109/TII.2021.3067026.

10. K. Wang, Y. Wang, Y. Sun, S. Guo, and J. Wu, “Green Industrial Internet of Things Architecture: An Energy-Efficient Perspective,” IEEE communications magazine, vol. 54, no. 12, pp. 48–54, 2016, doi: 10.1109/MCOM.2016.1600399CM.

11. Cheryl Tan, "Companies looking to offset carbon tax will have to wait some time more for worthy projects," The Straits Times, Jan, 2024. Available:

https://www.straitstimes.com/singapore/companies-looking-to-offset-carbon-tax-will-have-to-wait-some-time-more-for-worthy-projects

12. S. Zhu, K. Ota, and M. Dong, “Green AI for IIoT: Energy Efficient Intelligent Edge Computing for Industrial Internet of Things,” IEEE transactions on green communications and networking., vol. 6, no. 1, pp. 79–88, 2022, doi: 10.1109/TGCN.2021.3100622.

13. K. Dhondge, R. Shorey, and J. Tew, “HOLA: Heuristic and opportunistic link selection algorithm for energy efficiency in Industrial Internet of Things (IIoT) systems,” in COMSNETS : 2016 8th International Conference on Communication Systems and Networks : 5-10 January 2016., 2016, pp. 1–6. doi: 10.1109/COMSNETS.2016.7439960.

14. D. Swaminathan, et al., "ODTRA-Based Task Offload Optimisation For IIoT Systems: Improving Efficiency And Performance With Digital Twins And Metaheuristic Optimisation," IEEE Access, vol. 12, pp. 51796-51817, 2024.

15. R. W. L. Coutinho and A. Boukerche, “Modeling and analysis of a shared edge caching system for connected cars and industrial iot-based applications,” IEEE Trans. Ind. Inform., vol. 16, no. 3, pp. 2003–2012, 2020

16. T. Olofsson, A. Ahlen, and M. Gidlund, “Modeling of the fading statis- ´ tics of wireless sensor network channels in industrial environments,” IEEE Trans. Signal Process., vol. 64, no. 12, pp. 3021–3034, 2016.

17. F. Pantisano, M. Bennis, W. Saad, and M. Debbah, “Cache-aware user association in backhaul-constrained small cell networks,” in Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2014 12th International Symposium on, 2014, pp. 1–42. doi: 10.1109/WIOPT.2014.6850276.

18. L. A. Adamic and B. A. Huberman, “Zipf’s law and the internet,” Glot-tometrics, vol. 3, pp. 143–150, 2002.

19. C. Bernardini, T. Silverston, and O. Festor, “MPC: Popularity-based caching strategy for content centric networks,” in 2013 IEEE International Conference on Communications (ICC), 2013, pp. 3619–3623. doi: 10.1109/ICC.2013.6655114.

20. J. Park, P. Popovski, and O. Simeone, “Minimizing Latency to Support VR Social Interactions Over Wireless Cellular Systems via Bandwidth Allocation,” IEEE wireless communications letters., vol. 7, no. 5, pp. 776–779, 2018, doi: 10.1109/LWC.2018.2823761.

21. Moez Ali, “Handling Machine Learning Categorical Data with Python Tutorial”, DataCamp, Feb, 2023. Available: https://www.datacamp.com/tutorial/categorical-data [Accessed: 24-Mar-2024]

22. I. Bilbao and J. Bilbao, “Overfitting problem and the over-training in the era of data: Particularly for Artificial Neural Networks,” in *ICICIS : 2017 eighth*

International Conference on Intelligent Computing and Information Systems : 5-7 December 2017., 2017, pp. 1929–177. doi: 10.1109/INTELCIS.2017.8260032.

23. S. Ö. Arik and T. Pfister, "TabNet: Attentive Interpretable Tabular Learning," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 8, pp. 6679-6687, May 2021.