PRACTICE SCHOOL I
FINAL REPORT

QUANTUM MACHINE LEARNING APPLICATIONS ON ESCAPE ROUTING PROBLEMS

By

Parameswaran Iyer, Jetain Chetan and A Vinil

Prepared in partial fulfillment of
BITS F221 Practice School - I

At

Centre for Development of Advanced Computing, Silchar, Assam

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

June, 2024

Acknowledgement

We would like to thank everyone who has guided us throughout this project. Firstly, we would like to thank our project mentor, Mr.Nagendra Singh, whose valuable insights and encouragement, have been instrumental in the progress of this project. We are also grateful to the PS-1 supervisor, Prof.V Satya Narayana Murthy, for his continuous support, and prompt response to any logistical difficulties we faced.

We extend our sincere thanks to CDAC, CINE for providing the platform and resources necessary for this project. We are also thankful to BITS-Pilani for offering us the opportunity to engage in this enriching academic endeavor.

Thank you all for your invaluable contributions and unwavering support.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

Practice School Division

Station: Centre for Development of Advanced Computing **Centre:** Silchar,Assam
Duration: 28th May 2024 - 23rd July 2024 **Date of Submission:** 21st June 2024

Title: Quantum Machine Learning Applications on Escape Routing Problems

Authors:

Name	I.D. No.	Discipline
Parameswaran Iyer	2022AAB40415H	Electronics & Communication and Mathematics
Jetain Chetan	2022AAPS1551H	Electronics & Communication
A Vinil	2022A3PS1648H	Electrical & Electronics

Name of Expert	Designation
Mr. Nagendra Singh	Scientist B

Name of PS Faculty	Designation
Dr. V Satya Narayana Murthy	Associate Professor

Project Areas: Quantum Computing, Shortest Path Algorithms, Machine Learning

Abstract

This project aims to study a hybrid quantum machine learning system that can be used as an efficient and real-time implementable solution to assist in evacuation during natural disasters, particularly earthquakes. During natural disasters, haphazard evacuation without planning can lead to unnecessary damage to life and property. A centralized system that provides effective and efficient solutions for citizens is extremely necessary in disaster-prone locations.

One of the challenges lies in dynamic computation complexity and the real-time availability of reliable data on the effects of an earthquake. A parallel hybrid network(PHN) combining the advantages of classical and quantum computation is being investigated to solve the computational complexity problem. The study underscores the importance of Graph Neural Networks as an alternative model for solving this problem.

Contents

1 | Introduction

1.1 | Quantum Computing

- Quantum Computing(QC) is an area at the intersection of Computer Science and Physics. It is a completely different approach as opposed to classical methods of computing. QC leverages the fundamental principles of Quantum Mechanics: Entanglement and Superposition to perform quantum computations. It is proposed that Quantum Computer will help us solve problems currently intractable or unsolvable by classical computers. As Feynman said:
  Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical.

- Thus one of the aims of QC is to simulate quantum systems like molecules. On the other hand, scientists are trying to find Quantum solutions to problems that will provide an exponential speedup over the classical solutions. Additionally, quantum computers are being developed to find solutions to problems beyond the reach of most powerful supercomputers one day. This milestone is called the Quantum Advantage. One example of quantum advantage is finding prime factors of integers using Shor's algorithm on QCs in sub-exponential time complexity. Quantum Machine Learning(QML) is another such application where the processing capabilities of a QC can be leveraged to handle a large amount of data and solve complex mathematical problems in real-time.

1.2 | Quantum Machine Learning

- Quantum Machine Learning(QML) is a cutting-edge research area that aims to use Quantum Computing to solve machine learning problems. The capabilities of machine learning algorithms are fundamentally tied to the hardware they run on. The success of modern deep learning, for example, relies heavily on the parallel processing power of GPU clusters. QML extends this pool of hardware for complex machine-learning algorithms to QCs. From the modern viewpoint, QCs can be used and trained like neural networks.

- This project studies an innovative application of a hybrid quantum machine learning model for escape routing and evacuation mapping. Further, we propose graph neural networks as an alternative solution to this problem and highlight that further work needs to be done to find a quantum graph neural network solution to this problem.

1.3 | Escape Routing

- Escape Routing is a class of problems in graph theory pertaining to finding optimal paths in a graph between two nodes. Escape routing algorithms play an important role in modern-day industry. They are used in finding optimal routing on Printed Circuit

Boards (PCBs), path planning in autonomous vehicles, enhancing disaster management through large-scale evacuation planning etc.

- Disaster management teams leverage escape routing algorithms for large-scale evacuation planning in emergencies like floods or earthquakes. By analyzing population density, terrain, and infrastructure damage, these algorithms predict potential bottlenecks and suggest optimal evacuation routes for entire communities, potentially saving lives during critical situations.

1.4 | QML in Escape Routing

- Escape routing algorithms on large graphs like those in disaster management and evacuation problems require high computational power to be realizable. Furthermore, dynamically updating graphs require multiple iterations to accommodate the variation in parameters and output new optimal paths in real-time. The requirement for higher computational speed only increases with the scale of application.
- QML offers an innovative solution to this problem. By applying the principles of quantum mechanics, we can exponentially increase speed of computation. Further, a hybrid network consisting of both quantum and classical neural networks can be used to compute escape routing algorithms to give results of similar accuracy and optimal nature as those given by classical algorithms like Dijkstra's or A* algorithms.

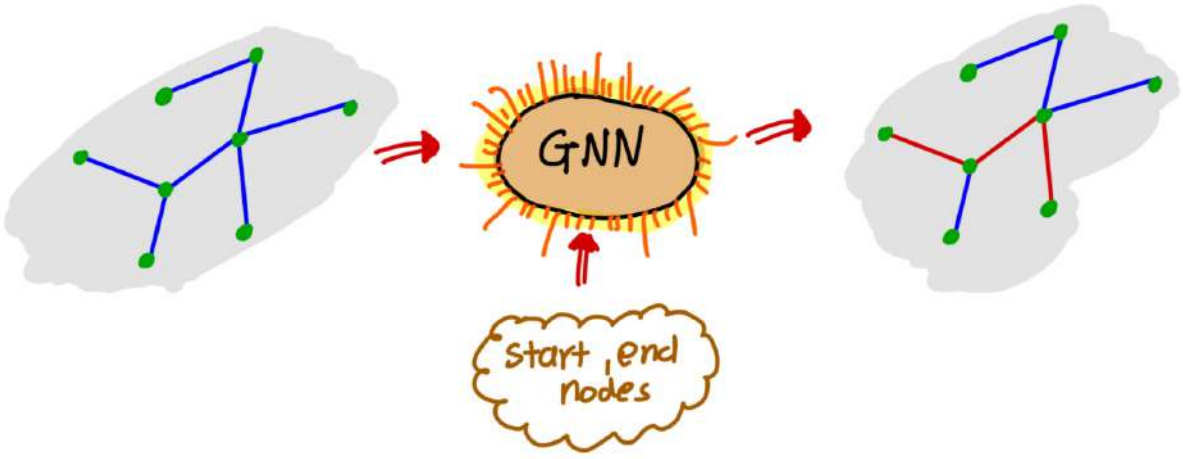


Figure 1.1: Desired GNN model: Input graph(left), Optimal path(right) *(Credit: Vinil)*

1.5 | Graph Neural Networks

- Graph Neural Networks (GNNs) are a class of deep learning methods designed to perform inference on data described by graphs [11]. A graph consists of nodes and

edges connecting the nodes. The nodes and edges both can have features describing them. For instance, a graph of a social network can have people as nodes with their name, age, and sex as features while the edges connecting them to other people can have features telling how they are related. Similarly, there can be a graph of a city with the nodes depicting various places and the edges, the roads connecting them. What our research further aims to do is the following:

- This project deals with the application of GNNs in computing the most optimal path between two nodes on a given graph. where nodes represent intersections and edges represent the roads connecting these intersections. Suggested GNNs such as [11] propose solutions whose time complexity does not favor shortest path computations for large networks. Hence the team aims to conduct a further investigation into the possibility of a Quantum GNN to solve this problem.

2 | The Project Overview

The project *QML Applications on Escape Routing Problems* gave the team a hands-on experience in Machine Learning(ML) and Quantum Computing. The area of the study presented a valuable opportunity to expand our knowledge and develop new skills. This project offered the team a platform better to comprehend the rapidly developing fields of QC and ML. In alignment with the same goal, we conducted a comprehensive literature review, exploring a wide range of resources, that provided valuable insights into various aspects of the task at hand.
The team was provided with the problem statement over the initial discussions with the mentor. Broadly, it consisted of two aspects:

1. Prediction of the most optimal path between a random starting node on a dynamic city graph and one of the predetermined exit nodes using a Classical ML model.

2. Investigation of a Quantum Machine Learning solution to this problem to further optimize the implementation.

Given the inherent complexity and vastness of the topic, the team decided to narrow the scope initially to solve the problem layer by layer. This involved focusing on a well-defined sub-problem: Prediction of the shortest path between two random nodes in a static graph using A* and later a Graph Neural Network.

During the initial phase of the project, we analyzed the primary material [1] in-depth, drawing inferences, and discussing various approaches to the same. We started with implementing the A* algorithm for single source shortest path computation on static graphs (C++ and Python implementations in Appendix A), as well as code written by Jetain to convert static graphs to dynamic graphs based on the model described in the paper. This was primarily aimed at understanding various aspects of graph theory that were needed for the project. Over a month, the team reviewed various papers and open-source material and learned concepts associated with them. An abridged version of our notes is mentioned in Appendix C.

Following the initial stage of grasping core concepts, the team worked on implementing various sub-parts of the main block. While initial impressions suggested a straightforward task, the complexity of training a graph using machine learning techniques quickly became apparent. Classical machine learning models, using linear regression or simple classification, proved insufficient due to the complexity of graphs. The team started the process by constructing basic models and training them on datasets generated through independent executions of Dijkstra's algorithm. However, graph training necessitated a more nuanced approach.

Due to the apparent limitations of generalizability, Vinil undertook a comprehensive review of potential models and identified the potential of Graph Neural Networks (GNNs) in the latter stage of the project. Vinil spent weeks searching for potential models to use. Reference [10] provided a valuable implementation of this architecture. Vinil further adapted the code

to incorporate a graph representation of a city generated using the OSMnx library (Appendix B) - a collaborative effort by Parameswaran, Jetain, and Vinil.

3 | Project Roadmap

1. **Study and Analyze the Research Paper** [1]
   Gain a thorough understanding of the concepts and methodologies presented.

2. **Implement A* Algorithm in C++ and Python**
   Develop the A* algorithm in both C++ and Python (see Appendix A).

3. **Summarize Understanding of** [1]
   Compile a summary that encapsulates the key insights from the paper.

4. **Find Alternative Solutions to the Problem**
   Explore and identify different approaches to address the problem.

5. **Study Graph Neural Networks**
   Dive into the principles and applications of Graph Neural Networks (GNNs).

6. **Build a Sample GNN Model Based on** [10]
   Construct a basic GNN model inspired by the methodologies in [10].

7. **Develop GNN for Dynamic Graphs:**
   The data extracted and processed needs to be passed into a temporal graph network [13] with the update functions in [1] in the memory unit of the TGN.

4 | Progress Till Midsem

The team has understood the fundamentals of quantum computing, including the mathematical basis, applications, and relevance to the project. This was a pre-requisite to understanding the given material to the depth required for this project. Further, the team has understood the graph algorithms that are used in optimal path detection, with a primary focus on Dijkstra's shortest path and A* algorithms.

The team has also studied additional material on dynamically evolving graphs and is working towards simulating the same. The team has also implemented A* and Dijkstra's algorithms in C++ and Python available in Appendix A. The first week of PS-I was utilized to create teams and provide orientation towards the respective projects. The next 2 weeks of the project were spent understanding the fundamentals required for the project. The team has analyzed the primary reference material and submitted detailed reports of the same to the mentor-in-charge in parallel.

The team also reviewed the paper and came up with a few critiques and oversights regarding the data used to train the model. The paper is not based on data obtained from real-life disasters, but rather on a mathematically ideal representation of an earthquake. Whether the simulated modeling of the earthquake the paper has provided is viable to apply to real earthquakes is debatable because the simulation doesn't take into account several factors like real-time traffic conditions and elevation. The team aims to provide solutions to these problems and implement the proposed architecture as a whole. The paper also chose to model earthquakes while neglecting auxiliary complexities that arise from an earthquake like landslides and the unusability of roads. Currently, the paper uses arbitrary linearly assigned values of edge weights to model all the effects of an earthquake. The team aims to change this by developing a better mathematical model to approximate this multivariate system.

Finally, over the 4th week of PS-I, the team has developed code to simulate dynamic graphs. This includes converting OpenStreetMap [6] data into graphs with node edges and weights and replicating the graph's dynamic behavior by adjusting edges' weights as time increases. This processing is done using the Python library OSMnx [3] using the following code snippet in Appendix B. The city of Guwahati which is prone to earthquakes was used for the same as shown in Figure 4.1.

(a) Map of Guwahati obtained from OpenStreetMap

(b) Graph representation of Guwahati

Figure 4.1: Visualization of Guwahati as a graph

5 | Review of Paper[1]

Here is briefly review the paper on escape routing[1].
An emergency due to a natural disaster, such as an earthquake, can be modeled as a dynamically evolving graph, where the shortest path must be found from a random starting point to established and fixed exit points. Existing graph algorithms, such as Dijkstra's Shortest Path Finding algorithm, work well on static graphs. However, modifications are necessary to extend its functionality to dynamic domains, leading to the development of the node-wise Dijkstra's algorithm.

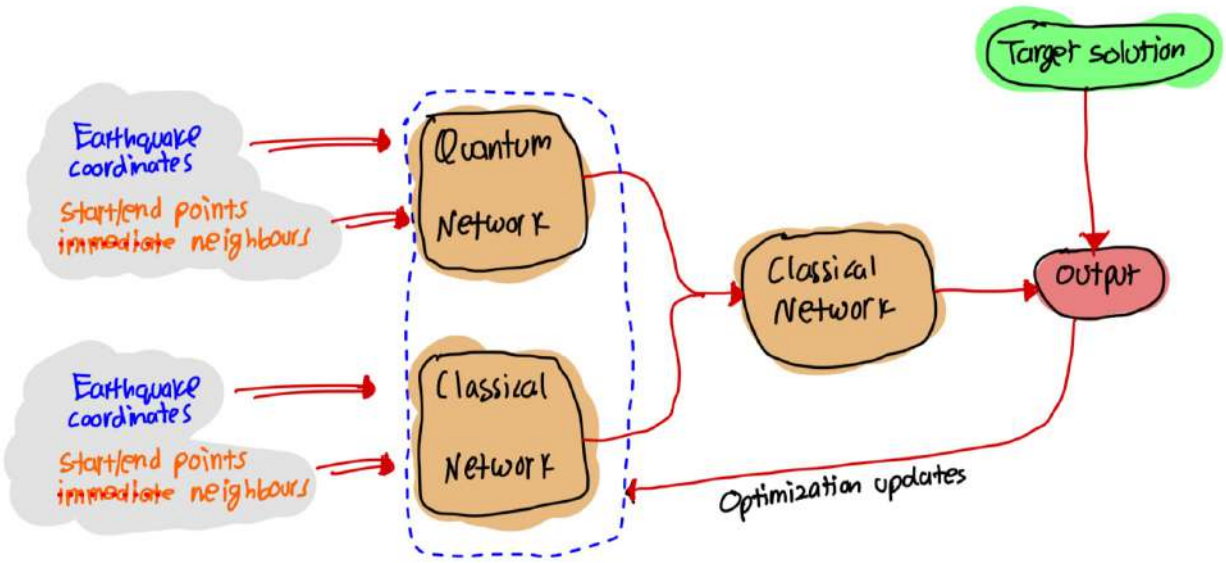


Figure 5.1: Flowchart of Hybrid Model *(Credit: Vinil)*

Furthermore, the results from node-wise Dijkstra's algorithm are used to train a hybrid quantum neural network(QNN). This network utilizes a novel quantum feature-wise linear modulation (FiLM) network parallel to a classical FiLM network, aiming to imitate the node-wise Dijkstra's algorithm while only accessing a reduced portion of the graph.

The proposed circuit architecture consists of a Parallel Hybrid Network (PHN) model integrated with a Feature-wise Linear Modulation (FiLM) model. This composite model features both a classical FiLM neural network and a quantum neural network. The inputs to this model include earthquake coordinates, start and end points, and information about the immediate neighbors of the nodes. Both the classical and quantum networks process these inputs. After processing, the outputs are combined linearly to produce an outcome, which is then passed to a fully connected layer and reduced to five values.

These five values act as a logit layer (raw output) for the node classifier, and the neighboring node corresponding to the highest number is chosen as the next node. This model effectively determines the subsequent node for an optimal path solution. Owing to the critical nature of efficient escape routing and evacuation mechanisms in times of emergencies, it is essential to explore, study, and develop methods to improve existing infrastructure to yield faster results.

Quantum Computing provides a fascinating avenue for research, poised on the brink of a technological revolution. Given that QC is faster and more efficient than classical computing but is still a developing field in terms of available hardware, this solution aims to leverage hybrid models, combining both quantum and classical machine learning, to arrive at solutions to the escape routing problem.

6 | Progress Post Midsem

Post mid-semester evaluative, the team shifted its attention towards studying graph neural networks. We read the paper [10] on GNN that finds the shortest path between two points in a static graph. After reviewing it, the team decided to modify the code provided in the paper to suit the problem in hand. In the code, the author used sample adjacency and feature matrices for demonstration purposes. The team replaced that with our adjacency and feature matrices generated by a graph using OSMnx and NetworkX libraries.

Initial code implementations encountered debugging challenges, requiring several days to resolve seemingly simple errors. Despite extensive efforts, the code's functionality was limited to graphs with a specific node count (less than or equal to 25). While a larger real-world graph (Guwahati with 5102 nodes) was attempted, the code encountered limitations. As a result, we opted for a smaller test graph (Opatowiec with 25 nodes) to demonstrate core functionalities. This experience highlights the need for further code optimization and scalability improvements for handling larger datasets. Apart from that, we tried to implement the hybrid quantum-classical model given in the paper[1]. There we embedded data, generated by running Dijkstra's algorithm, into Quantum states using CNOT gates and Rotational gates like RZ. The CNOT gate was used to create entanglement between qubits. And the rotational RZ gates were used to map classical data to quantum data. For instance, let's say we have a list of numbers 1,2,3. Now, we feed this to the RZ gate which then maps it to some specific angles, say 1 to 45 degrees, 2 to 47 degrees etc. Working in this quantum state of angles makes it easy to manipulate large number of data points simultaneously.

The team further studies Temporal Graph Networks (TGN)[13] to model the GNNs for dynamically updating graph as required by the problem statement. The model was originally developed for the messaging platform Twitter to handle dynamically updating features of a network of connections. The code was further modified to suit our needs by changing the memory function to the update functions mentioned in the primary reference[1]. The team is currently working on improving this model and debugging various errors while trying to improve time complexity of operations.

Quantum Computing (QC) holds promise for efficiently tackling large-scale graph problems like optimal pathfinding. Quantum Graph Neural Networks (QGNNs) emerge as a promising research avenue with the potential to overcome limitations faced by classical networks in handling complex, large-scale graphs. QGNNs, by learning correlations at various levels (node, subgraph, whole-graph), could pave the way for solving intricate routing problems, potentially extending to real-time navigation scenarios. As societal networks become denser, the ability to model problems as graphs and leverage graph machine learning for efficient solutions will be increasingly valuable.

7 | Code Explanation

Final Model : Code

7.1 | Generating Graph

The python OSMnx package is used to obtain the graph of "Opatowiec" from OpenStreetMaps. This graph is further processed by making it undirected, weighted and adding the features of travel time and speed to each edge. This is the graph that will be used to run the GNN for the remaining code.

7.2 | Data Generation

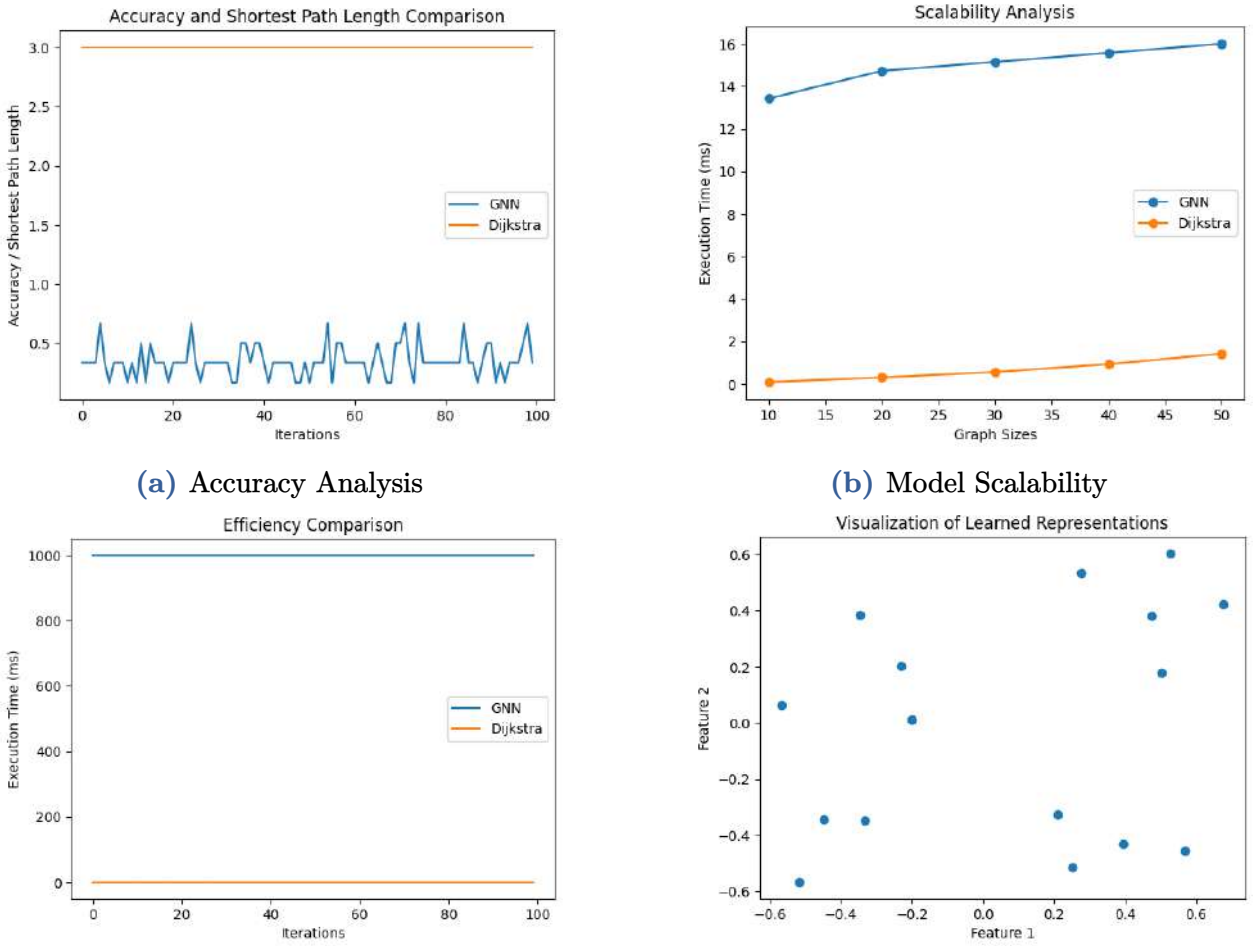
Section 2 deals with iterating through every node pair in the graph of Opatowiec and calculating the least cost path between each node pair using the *shortest_path*() function of OSMnx. The data is then converted into required format and saved as a CSV file to be used to train the GNN. Considering the low number of nodes in the map, the entire data set is used to avoid underfitting.

7.3 | Code from Paper[1]

This section is an implementation of the method described in the primary reference material. A GNN class is defined and the behavior of the GNN is described to find the optimal path between any two nodes. The model is trained and Scalability and Accuracy analyses is conducted to analyze the efficiency of the GNN model described in the reference material. The results of the same are in the figure 7.1.

7.4 | Creating the GNN Model

An adjacency matrix is generated from the graph data as it is the input format to the GNNs. Various features of the graphs such as nodes, edges, coordinates of nodes, length of edges etc are generated and converted into dataframe format as required by the GNN input format. Further catagorical features are encoded and processed into numerical formats. The numerical data are normalized. The Model is trained on the graph and a random set of pair of nodes are genrated and used to test the model. The analysis of the same is depicted in figure 7.2



(a) Accuracy Analysis

(b) Model Scalability

(c) Model Efficiency

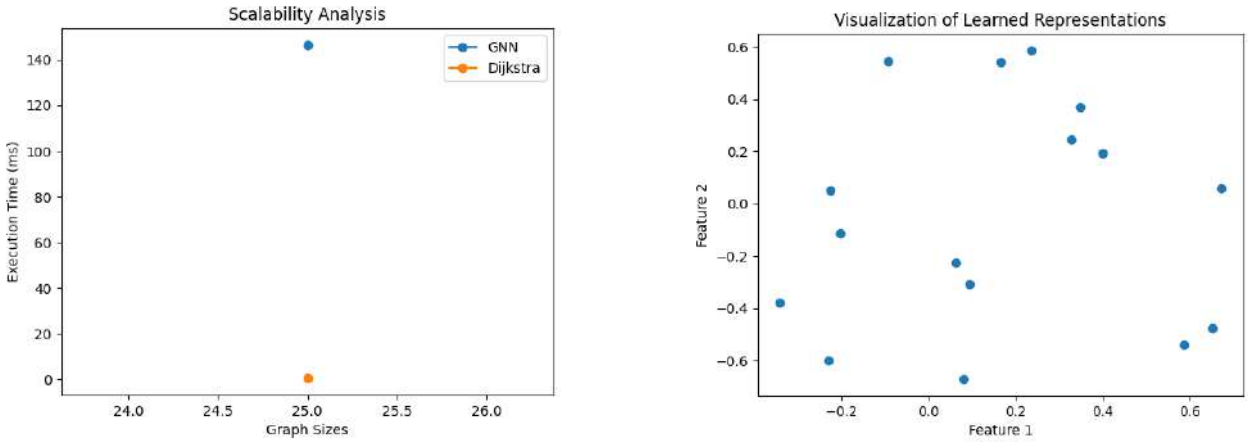
(d) Visualization of learned represenation

Figure 7.1: Analysis of model described in references

8 | Conclusion

In summary, the exploration of Quantum Machine Learning algorithms for escape routing and evacuation mapping, as discussed in the paper by Nathan Haboury et al., demonstrates a significant advancement in addressing the challenges posed by dynamically changing environments during natural disasters. By leveraging a hybrid model that integrates both quantum and classical neural networks, this approach offers a promising solution for optimizing escape routes in real time. This innovative method underscores the critical need for continued research and development in computational techniques to enhance emergency response and preparedness.

According to the analysis done by the authors, the model seems to be promising. However, the authors have tested it on only a small graph, and graphs of larger sizes need testing to check for accuracy and time constraints. The authors also suggest a reinforcement model instead of a supervised setting as an avenue for further exploration. The analysis concluded that the model could learn to match Dijkstra's optimally with only local information, making



(a) Model Scalability **(b)** Visualization of learned represenation

Figure 7.2: Analysis of designed model

it computationally workable in an evolving situation. The hybrid model seems to have 7% more accuracy than just the classical approach.

We suggest an alternative approach to tackling this problem of finding the most optimal path between two points in a graph of a city: Graph Neural Networks. GNNs are at the research frontier of machine learning right now. We studied the paper [10] that used a GNN model to find the shortest path in a static graph. Further work needs to be done on large graphs with hundred thousand nodes. We carried out a literature review of the paper [10] and submitted our results to our mentors. The document can be found in Appendix D.

Finally, we conclude that the project was an enriching experience. It introduced us to the world of machine learning and quantum computing. We found immense joy in being able to understand a scientific paper and trying to implement it. We also gained skills in writing a technical report, presenting our work to the mentors, and engaging in fruitful group discussions with our Professor. We would like to continue our research in the area of Graph Machine Learning and contribute to the field by innovating new models that constantly challenge the status quo.

9 | References

1. Haboury, N., Kordzanganeh, M., Schmitt, S., Joshi, A., Tokarev, I., Abdallah, L., Kurkin, A., Kyriacou, B., and Melnikov, A. (2023, July 28). A supervised hybrid quantum machine learning solution to the emergency escape routing problem. arXiv.org. https://arxiv.org/abs/2307.15682

2. IBM Quantum Learning. (n.d.). Retrieved June 21, 2024, from https://learning.quantum.ibm.com/

3. OSMnx Documentation. (n.d.). Retrieved June 21, 2024, from https://osmnx.readthedocs.io/en/stable/

4. C. Bernhardt, *Quantum Computing for Everyone*. The MIT Press eBooks, 2019. DOI: https://doi.org/10.7551/mitpress/11860.001.0001

5. T. Dai, W. Zheng, J. Sun, C. Ji, T. Zhou, M. Li, W. Hu, and Z. Yu, "Continuous Route Planning over a Dynamic Graph in Real-Time," *Procedia Computer Science*, vol. 174, pp. 111–114, 2020. DOI: https://doi.org/10.1016/j.procs.2020.06.065

6. OpenStreetMap. (n.d.). OpenStreetMap. Retrieved June 21, 2024, from https://www.openstreetmap.org/#map=13/26.1522/91.7230&layers=CG

7. I. Quilez, "The Hidden Beauty of the A* Algorithm," YouTube, Jan. 21, 2023. Retrieved June 21, 2024, from https://youtu.be/A60q6dcoCjw

8. Gate Smashers, "A* Algorithm in AI (Artificial Intelligence) in HINDI | A* Algorithm with Example," YouTube, April 5, 2019. Retrieved June 21, 2024, from https://youtu.be/tvAh0JZF2YE

9. SP Pune, "VTU AIML LAB1 ASTAR || A* Search Algorithm," YouTube, Dec. 8, 2022. Retrieved June 21, 2024, from https://youtu.be/64q7zokfdJo

10. B. S. Neyigapula, "Graph Neural Networks for Optimal Pathfinding: Uncovering the Shortest Distances," Jawaharlal Nehru Technological University. [Online]. Available: https://bit.ly/3Wc1QBg

11. Graph Neural Network and Some of GNN Applications: Everything You Need to Know. [Online]. Available: https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications. [Accessed:18-Jul-2024].

12. Adjacency Matrix: Available: https://mathworld.wolfram.com/AdjacencyMatrix.html

13. Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., Bronstein, M. (2020, June 18). Temporal Graph Networks for Deep Learning on Dynamic Graphs. arXiv.org. https://arxiv.org/abs/2006.10637

10 | Appendix A

A* Algorithm implementation using C++:

```cpp
#include<bits/stdc++.h>
using namespace std;

#define INF 1e9 // Define a large value for infinity
int ans = 0; // Initialize the answer variable
typedef pair<int, int> ii; // Define a pair of integers as ii
typedef vector<ii> vii; // Define a vector of pairs as vii
typedef vector<int> vi; // Define a vector of integers as vi

#define LSone(j) ((j) & -(j)) // Macro to get the least significant bit

vector<vii> AL(9000); // Adjacency list for the graph
int d[9000]; // Array to store heuristic distances
int n, m, s, e; // Variables for number of nodes, edges, start and end nodes

// Function to read the graph input
void getgraph()
{
    int x, y;
    cin >> x >> y; // Read exit coordinates (goal position)
    d[e] = 0; // Set heuristic distance of the end node to 0

    // Read coordinates of other nodes and compute heuristic distances
    for (int i = 0; i < n; ++i)
    {
        if (i == e) continue; // Skip the end node
        int a, b;
        cin >> a >> b;
        d[i] = sqrt((a - x) * (a - x) + (b - y) * (b - y)); // Euclidean Distance
    }

    // Read edges and construct the adjacency list
    for (int i = 0; i < m; ++i)
    {
        int u, v, w;
        cin >> u >> v >> w;
        AL[u].push_back({w, v});
        AL[v].push_back({w, u});
    }
    return;
```

```
}

int main()
{
    cin >> n >> m >> s >> e;
    // Read the number of nodes, edges, start node, and end node

    getgraph(); // Call the function to read the graph input

    vi dist(n, INF); // Initialize distances to all nodes as infinity
    dist[s] = 0; // Distance to the start node is 0
    set<ii> pq; // Priority queue to store nodes with their distances

    // Insert all nodes into the priority queue with their heuristic distances
    for (int u = 0; u < n; u++)
        pq.emplace(dist[u] + d[u], u);

    // Main loop to find the shortest path
    while (!pq.empty())
    {
        auto [dh, u] = *pq.begin(); // Get the node with the smallest distance
        pq.erase(pq.begin()); // Remove the node from the priority queue

        // Relaxation step
        for (auto &[w, v] : AL[u])
        {
            if (dist[u] + w >= dist[v]) continue; // If no shorter path is found, continue
            pq.erase(pq.find({dist[v] + d[v], v})); // Remove the node from the priority queue
            dist[v] = dist[u] + w; // Update the distance to the node
            pq.emplace(dist[v] + d[v], v); // Insert the node back into the priority queue with the new distance
        }
    }

    return 0;
}
```

A* Algorithm implementation using Python:

```
# Define the graph using an adjacency list
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)]
}

# Function to get the neighbors of a given node
def get_neighbours(v):
    return Graph_nodes.get(v, [])

# Heuristic function to estimate the cost from a node to the goal
def h(n):
    H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]

# A* algorithm to find the shortest path from start_node to
   stop_node
def aStarAlgo(start_node, stop_node):
    # Initialize the open and closed sets
    open_set = set([start_node])
    closed_set = set()

    # Initialize the g and parents dictionaries
    g = {}
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node

```

```
    # Main loop of the A* algorithm
    while len(open_set) > 0:
        # Find the node with the lowest f(n) = g(n) + h(n)
        n = None
        for v in open_set:
            if n is None or g[v] + h(v) < g[n] + h(n):
                n = v

        # If no node is found, the path does not exist
        if n is None:
            print('Path does not exist!')
            return None

        # If the goal node is reached, reconstruct the path
        if n == stop_node:
            path = []

            while parents[n] != n:
                path.append(n)
                n = parents[n]

            path.append(start_node)
            path.reverse()

            print('Path found: {}'.format(path))
            return path

        # Loop through the neighbors of the current node
        for (m, weight) in get_neighbours(n):
            if m not in open_set and m not in closed_set:
                open_set.add(m)
                parents[m] = n
                g[m] = g[n] + weight
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n
                    if m in closed_set:
                        closed_set.remove(m)
                        open_set.add(m)

        # Move the current node from open set to closed set
        open_set.remove(n)
        closed_set.add(n)

    # If the open set is empty and the goal is not reached, the path
    does not exist
    print('Path does not exist!')
```

```
93      return None
94
95  # Example usage:
96  aStarAlgo('A', 'J')
97  # Output: ['A', 'F', 'G', 'I', 'J']
```

Code to convert a static graph to the dynamic graph described by the model:

```
from math import sqrt

class graph:
  def __init__(self,AdjList,Coord,EQcoord,exitNodes,startNode):
    self.AdjList = AdjList #AdjList[u] = [[v1,w1],[v2,w2],...]
    self.Coord = Coord #Coord[u] = (x,y)
    self.EQcoord = EQcoord #(x,y)
    self.exitNodes = exitNodes #ExitNodes[1/2/3] = [u1,u2,u3...]
    self.time = 0
    self.R_epi = 0.5 # formula is 0.5 + (0.0002*t)**0.5
    self.R_exit = 0 # formula is (0.00075)**0.5, three values for three exit nodes
    self.currNode = startNode # u
    self.currCost = 0

  def getAdjNodes(self):
    return self.AdjList[self.currNode]

  def travel_to_node(self,v):
    ## v is expected to be 0,1,2,3,4
    '''if u not in [self.AdjList[self.currNode][i][0] for i in range (len(self.AdjList[self.currNode]))]:
      print("Invalid node") ## May not be a necessary check'''

    self.ongoing_EQ_Effect() ## update according to EQ effect
    for exitNode in self.exitNodes: ## update according to traffic effect at each exit node
      self.ongoing_Traffic_Effect(exitNode)

    self.currCost += self.AdjList[self.currNode][v][1] ##Travelling to node
    self.currNode = self.AdjList[self.currNode][v][0]

    self.time += 1 ## updating time and radii of effects
    self.R_epi = 0.5 + sqrt(0.0002*self.time)
    self.R_exit = sqrt(0.00075*self.time)

  def get_D_epi(self,u): ## calc distance from epicenter
    temp = (self.Coord[u][0]-self.EQcoord[0])**2 + (self.Coord[u][1]-self.EQcoord[1])**2
    return sqrt(temp)

  def get_D_exit(self,u,exitNode):
    temp = (self.Coord[u][0]-self.Coord[exitNode][0])**2 + (self.Coord[u][1]-self.Coord[exitNode][1])**2
    return sqrt(temp)
```

```
41
42  def init_EQ_effect(self):
43    for u in range(len(self.AdjList)): ## checks each node's dist
     from epicenter and updates weights
44
45      D_epi = get_D_epi(u)
46
47      if D_epi <= 0.3*self.R_epi:
48        w_bias = 5
49      elif D_epi > 0.3*self.R_epi and D_epi <= 0.75*self.R_epi:
50        w_bias = 2
51      elif D_epi > 0.75*self.R_epi and D_epi <= self.R_epi:
52        w_bias = 1.3
53      else:
54        w_bias = 1
55
56      for AdjNode in self.AdjList[u]:
57        AdjNode[1] *= w_bias
58
59  def ongoing_EQ_Effect(self):
60    for u in range(len(self.AdjList)):
61      D_epi = get_D_epi(u)
62
63      if (D_epi <= 0.3*self.R_epi):
64        w_bias = sqrt(0.003*self.time+1)
65        for AdjNode in self.AdjList[u]:
66          AdjNode[1] = min(AdjNode[1]*w_bias, 5)
67
68      elif (D_epi > 0.3*self.R_epi) and (D_epi <= 0.75*self.R_epi):
69        w_bias = sqrt(0.002*self.time+1)
70        for AdjNode in self.AdjList[u]:
71          AdjNode[1] = min(AdjNode[1]*w_bias, 4)
72
73      elif (D_epi > 0.75*self.R_epi) and (D_epi <= self.R_epi):
74        w_bias = sqrt(0.001*self.time+1)
75        for AdjNode in self.AdjList[u]:
76          AdjNode[1] = min(AdjNode[1]*w_bias, 3)
77
78      else:
79        pass
80
81  def ongoing_Traffic_Effect(self,exitNode):
82    for u in range(len(self.AdjList)):
83      D_exit = get_D_exit(u,exitNode)
84
85      if (D_exit <= 0.3*self.R_exit):
86        w_bias = sqrt(0.003*self.time+1)
87        for AdjNode in self.AdjList[u]:
```

```
88            AdjNode[1] = min(AdjNode[1]*w_bias, 5)
89
90        elif (D_exit > 0.3*self.R_exit) and (D_exit <= 0.75*self.
    R_exit):
91          w_bias = sqrt(0.002*self.time+1)
92          for AdjNode in self.AdjList[u]:
93            AdjNode[1] = min(AdjNode[1]*w_bias, 4)
94
95        elif (D_exit > 0.75*self.R_exit) and (D_exit <= self.R_exit):
96          w_bias = sqrt(0.001*self.time+1)
97          for AdjNode in self.AdjList[u]:
98            AdjNode[1] = min(AdjNode[1]*w_bias, 3)
99
100       else:
101         pass
```

11 | Appendix B

Code snippet to convert OpenStreetMap to graph:

```
import csv
import random
import math
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
import networkx as nx
from sklearn.model_selection import train_test_split
import folium

# Place name used to search on OpenStreetMap
place_name = "Opatowiec"

# Fetch street network
graph = ox.graph_from_place(place_name, network_type='drive')

# Convert graph to undirected
graph = graph.to_undirected()

# Adding edge lengths as weights
graph = ox.distance.add_edge_lengths(graph)

#Some additional draft changes
graph = ox.routing.add_edge_speeds(graph)
graph = ox.routing.add_edge_travel_times(graph)

#Display graph
print(graph)
ox.plot_graph(graph, node_size=3, node_color='blue', edge_color='#999999',
    bgcolor='white', show=True)
```

The graph is shown in Figure 9.1.

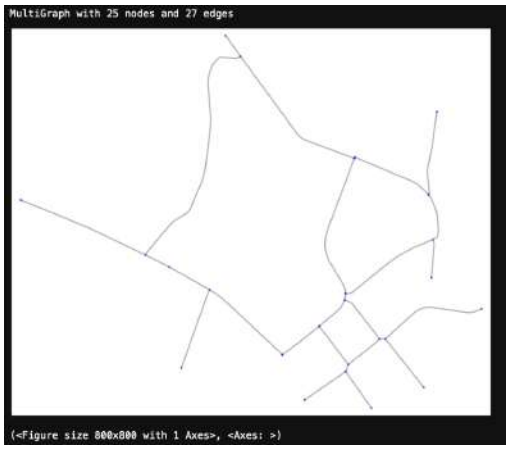


Figure 11.1: Graph of Opatowiec with 25 nodes and 27 edges

12 | Appendix C

NOTES AND OBSERVATIONS :

1. **Nathan Haboury et al.**[1]: Aims to find how supervised hybrid QML can optimize emergency evacuation plans for cars during natural disasters. The situation is modeled as a shortest-path problem on an uncertain and dynamically evolving map. Focusing on earthquakes, the objective is to obtain an optimal route for evacuating automobiles, hence minimizing travel time.

2. Proposes a new feature : Quantum feature-wise linear modulation(FiLM) neural network parallel to a classical FiLM network to imitate Dijkstra's node-wise shortest path algorithm on a deterministic dynamic graph. The term dynamic refers to major dynamic effects of an earthquake on the topography of a graph includingt land deformation, collapse of buildings or debris.

3. Comparison and Context of the choice of shortest path algorithms is in table 10.1

	Optimality	Performance
Dijkstra's	Guarantees optimality of shortest path	Explores all possible paths; hence computationally expensive for our purpose
A*	Optimality is conditional to the consistency of the heuristic.	Reduced search space leading to faster performance at the cost of accuracy. Building optimal heuristic function is key.

Table 12.1: A* vs Dijkstra's single source shortest path algorithms

4. The paper introduces a hybrid QML approach that requires only local information, as opposed to the global information required by classical Dijkstra's algorithm, to mimic the node-wise Dijkstra's algorithm in terms of path quality on a dynamic graph.

5. Model efficiency is done through ZX Calculus, Fourier embedding analysis, and Fisher expressivity:

   - **ZX Calculus**: A graphical language for quantum computing that simplifies the representation and manipulation of quantum operations.
   - **Fourier Embedding Analysis**: A technique that uses Fourier transforms to represent sequences in the frequency domain, capturing periodic and structural properties.
   - **Fisher Expressivity**: A measure of a model's expressive power based on Fisher information, indicating the model's sensitivity to changes in underlying parameters.

6. Python OSMnx package is used to convert any selected region of a map into a graph.

7. Steps followed in Hybdrid Quantum Neural Network(HQNN) Model :

   - Dataset is produced by simulating an earthquake at randomized coordinates in the city and then collecting routing data for each earthquake simulation.
   - Both the classical and quantum neural networks and fed with the following input features : Earthquake coordinates, start and end nodes and immediate neighbours.
   - The output of classical FiLM and quantum neural networks are combined to produce the outcome which is fed to a trainable layer.
   - The outcome of the trainable layer is 5 numbers that act as the logit(raw outputs) layer of the node classifier
   - Finally, the neighboring node corresponding to the highest number is chosen as the next node.

   This is illustrated in figure below :

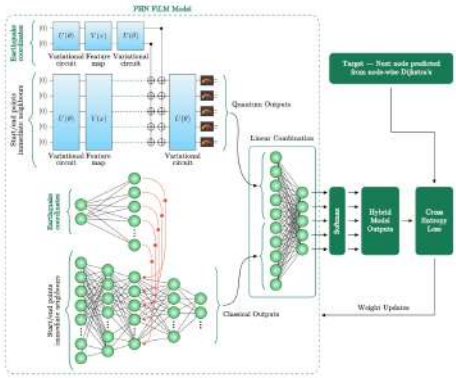


Figure 12.1: Hybrid Quantum Neural Network

8. **Results and Analysis**: Results show that HQNN performs better than the Classical NN. The runtime of HQNN only depends on the number of nodes in one path which in the worst-case scenario includes all nodes $O(n_{nodes})$.

9. **Practical Analysis**:

   - PHNs can suffer from primacy, where the network favors the output of either the MLP (Multilayer Perceptron) or the VQC (Variational Quantum Circuit), leading to suboptimal performance. To evaluate the contributions of each sub-network, the weights of the final layer (a 5×10 matrix) were analyzed. This analysis showed two key points:
     1) The weights are similar between the VQC and MLP, indicating no primacy
     2) The VQC weights show smoother transitions, while the MLP weights are more irregular.
     Relative contribution is quantified by the Frobenius norm.

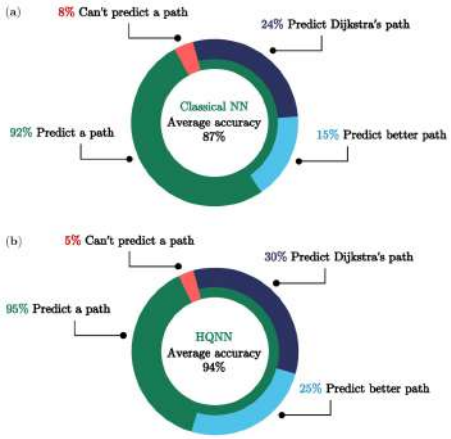


Figure 12.2: Results

- Performance on QPU: On a 25 qubit ion-based QC, with 3 decision points, the tasks took UB 10 mins including queuing effects. Comparison between VQC and QPU outputs qualitatively shows the high correlation between the simulator and actual hardware adjusting for shot noise and gate errors.

13 | Appendix D

WORK REQUEST FORM NOTES :

Problem Definition:
Predict the most optimal path between a random starting node on a dynamic city graph and one of the predetermined exit nodes using a Classical ML model. Is there a Quantum Machine Learning solution to this problem?
Given the complexity of the problem, we have reduced it to the following form:
Predict the shortest path between two random nodes in a static graph using a Graph Neural Network.
Summary of our solution and further steps:
We used an existing implementation of a finding graph neural network developed in this paper[10]. This method uses a sample adjacency matrix and feature matrix. However, we used OSMnx to convert a city map into a graph after which its adjacency and feature matrics were derived. The *adjacency matrix*, sometimes called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (vi,vj) according to whether vi and vj are adjacent[12]. A *feature matrix* in the context of graph neural networks (GNNs) is a matrix where each row corresponds to a node in the graph, and each column corresponds to a feature of the nodes. This matrix captures various attributes or properties of the nodes, such as position, weight, type, or any other relevant information that can help the GNN learn meaningful patterns. We used latitude and longitude as node features. The GNN is intended to learn the correlations between the nodes and thus help in finding the shortest path. Later, the GNN is compared with classical Dijkstra run on the same graphs, and their accuracy, scalability, and efficiency are compared. In the paper, the GNN is shown to have a superior performance over Dijkstra in terms of all metrics. It also declares this as a promising approach for larger graphs. We tried to implement the code with a larger graph consisting of 25+ nodes. However, we kept getting errors about handling too many indices. Our implementation error will be known only through an increased understanding of graph theory, GNN, and more practice in training machine learning algorithms. For now, we have got a basic understanding of Graph Theory and Machine Learning. Now the question of finding a Quantum solution to this problem remains to be dealt with. As we get a mathematical understanding of Quantum Graph Theory and Graph Machine Learning, then only will a fruitful solution be developed in the future.
A note for code(Appendix E):
We have tried to comment on every line in the code to explain its underlying motive. The details of the model may be found in the paper[1]. The code files are Jupyter Notebooks(.ipynb) showing the outputs of each code cell and the errors at the end. Model1.ipynb is our implementation and Original.ipynb is the original code of the paper.