NANYANG TECHNOLOGICAL UNIVERSITY

COVID-19 DIGITAL CONTACT TRACING

WITH SEQUENCE EMBEDDING

Li Bingzi

School of Computer Science and Engineering

2021

NANYANG TECHNOLOGICAL UNIVERSITY

SCSE20-0326

COVID-19 DIGITAL CONTACT TRACING

WITH SEQUENCE EMBEDDING

Submitted in Partial Fulfilment
for the Degree of Bachelor of Engineering (Computer Science)
of Nanyang Technological University

by

Li Bingzi

Abstract

This project aims to propose a novel digital contact tracing solution TracingwPrivacy with privacy preserved for users, and deliver a Proof of Concept for the solution. The current contact tracing apps and QR code solutions are either centralised systems which sends every users' data to the government database, or partially decentralised systems where the government maintain COVID-19 patients' and all their contacts' data. Different from the current solutions, this project proposes a completely decentralised system, where the government has access to embeddings of users' trajectories, not the original data. TracingwPrivacy addresses the rising concern of data privacy in digital contact tracing.

Firstly, existing embedding algorithms were researched, and SGT (Sequence Graph Embedding) was found suitable. Then a multi-agent simulation of mobile system was built since we could not take data from the real world. The simulation generated data, which were embedded by SGT and sent to a server. The server would compute the similarity between the embeddings and return similar ones upon the frontend's request.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Ast. Prof. Yu Han. Without his patience and guidance, this project would not be possible. I gained an invaluable lesson from designing and implementing PoC of the system. I would like to thank my supervisor for checking and give great advice in the process.

I also want to express appreciation to Chen Zichen who helped clarify and give great ideas for my project with patience.

I would like to thank Dr. Liu Zelei for giving an example of a multi-agent simulation. Without his help, the simulation would be much more difficult.

At last, I would like to appreciate my school, School of Computer Science and Engineering NTU for providing me the chance to do this project.

Table of Contents

Table of Figures

List of Tables

1. Introduction

1.1. Background

The Coronavirus pandemic which is caused by the SARS-CoV-2 virus (Covid-19), has been a global scale disaster. As reported by the WHO, until February, there were more than 100 million confirmed cases of COVID-19, including more than two million deaths[1].

Contact tracing has been one of the top priorities in controlling the spread of Covid-19. As the number of cases continued to surge and soon exceeded the capacity of public health services, most governments turned to digital contact tracing.
There are various existing or proposed digital contact tracing solutions. From the sensor technology used, they can be categorised into QR code and Big data based, GPS based or Bluetooth based systems. From the software architecture, they are either centralised or decentralised systems. In centralised systems, personal data are collected through the sensor technology chosen and send to a central server cluster, which is controlled by the government authority. In decentralised systems, the data are stored on individuals' devices. However, the current decentralised solutions are not complete decentralised since the government maintains a special database only for covid-19 patients and their close contacts.[2]

A typical example is Singapore's TraceTogether app, a Bluetooth based decentralised system. It does not require every user's data. However, the data of a infected user and his/her close contacts can still be seen by the government officials. If the virus affects a larger population in Singapore, the MOH can get more people's data[2,3]. Beside TraceTogether app, there is another QR based centralised system SafeEntry, which stores every single data entry in a server owned by the government[4]. There has been debate and negative news towards TraceTogether and SafeEntry, including tracing data made available to police[5].

The need of encoding the data itself while preserving its features leads to embedding solutions. While there has been lots of embedding algorithm proposed, few people tried to apply embedding in contact tracing. Therefore, in this project a PoC is developed to fill in the gap.

The idea is that the trajectory of a contract tracing app user during a certain time period *[T1, T2]* can be regarded as a sequence. A sequence is defined as a string of ordered discrete alphabets. An alphabet can be an event, a value or in our case, the location. The feature embedding of a sequence is the representation of the sequence in vector space[6]. Embeddings are not human readable. Therefore, a tracing app preserves total privacy if it only sends the embeddings to the central server. The feature of users' trajectories is preserved in vector space so similarities between them could be computed[7].

1.2. Scope and Objective

This project aims to propose a Singapore digital contact tracing solution with complete privacy preserved by combining existing sequence embedding function[7] and the current mobile app solution like SafeEntry, and develop a Proof of Concept(PoC) for it. The scope of the project includes developing a system consists of three components.

The first is a real-time multi-agent mobile simulation, which records user's trajectory and perform sequence embedding at client's side. Only the embeddings are sent to the central server, not data itself. This component is where the data comes from since we are not able to collect trajectory data from the real world. The second part is a server which computes the similarity between sequence embeddings upon the request from the frontend. Finally, a frontend that allows a MOH staff to search close contacts with a patient's sequence embedding.

In addition, there are pieces of code to how that the system can distinguish between similar and different trajectory pairs.

1.3. Potential Issues, Limitations and Challenges

1.3.1. Scaling

Scaling is an issue when building a real-world contact tracing app according to the PoC. As the number of nodes in the trajectory and the number of users grows, storage and query speed are two main issues. Embeddings need compression in data transmission and storage. The process of information retrieval needs to be optimized to ensure responsive query. The details are discussed in Chapter 7.

1.3.2. Effectiveness

In Chapter 5, the theory of SGT algorithm is introduced. An example proves that SGT embedding is able to represent the features of similar and different sequences. However, it remains unknown if the system is effective or not in the complex situation of virus spreading.

1.4. Report Outline

This report consists of five chapters. Chapter1, the current chapter, introduces the background, motivations, scope and objectives, the potential issues, limitations and challenges. Chapter 2 is the literature review. In this section, the current apps and solutions which are mentioned in the introduction are explained in detail. Chapter 3 describes the core algorithm (SGT) used in the PoC and the tools and packages used. Chapter 4 presents the project schedule and plan. Chapter 5 shows the methodologies and design documents. Chapter 6 shows the experimental results and discuss how the algorithm helps search close contacts of a patient. Chapter 7 draws the conclusion from all previous sections, and states possible future work recommendations.

2. Literature Review

2.1. Existing Digital Contact Tracing Solutions

2.1.1. Absolute Location vs Relative Location

In digital contact tracing systems, the goal is to find close contacts of the infected patient. The process to decide whether two users are close contact or not is often based on geo information. We can categorize location data into two kinds: relative location data, and absolute location data[3].

Typical example of absolute location data is GPS system. Relative location data can be collected through Bluetooth. If two smartphones with Bluetooth on gets into a certain range, it can be recorded down. Both absolute location data and relative location data is widely applied in digital contact tracing.

2.1.2. Centralized vs Decentralized Architecture

In centralized architecture, the software resides at a central location. In digital contact tracing, it refers to the data and the algorithm resides in a server or server cluster owned by the government. In decentralized architecture, the software resides on each client machine.[8]

2.1.3. TraceTogether and How It Works

There are plenty of existing contact tracing systems globally. This project proposes a solution for Singapore. Therefore, TraceTogether, the current solution in Singapore, is discussed in detail here.

The typical process of digital contacting tracing can be divided into four phases in a workflow[3]:

(1) Initialization phase: the system is set up by individuals and other parties.

(2) Sensing phase: the system records down each user's relevant data, relative location or absolute location.

(3) Reporting phase: an infected user reports his/her data to Ministry of Health.

(4) Tracing phase: a third party, such as MOH, will collect the infected user's data and search the close contact based on the data.

For TraceTogether, the following is the entire process of contact tracing:

(1) Initialization phase: the user installs the TraceTogether app on his/her smartphone. The app sends the user's phone number N to MoH. MoH server generates a pseudonym PID and store (N, PID) in the database. Then MoH encrypts the PID and a certain time interval T with secret key K, and sends INFO = Encryption(PID, T, K) to the user's smartphone.

(2) Sensing phase: In the time interval T, if two users i and j come into a certain Bluetooth range, they will store ($INFO_i$, $INFO_j$, Signal Strength) locally on their phones.

(3) Reporting phase: A user diagnosed with covid-19 have to give MoH all the locally stored pairs ($INFO_i$, $INFO_j$, Signal Strength).

(4) Tracing phase: MoH decrypts all the encrypted pairs and obtain PID of all users who came close with the patient. Then MoH obtain the phone number from the PID and contact them.[3]

2.2. Data Regulation and Technology Protocols

2.2.1. PDPA

Personal Data Protection Act (PDPA) is a framework that gives a baseline for data protection and privacy in Singapore. Similar to other personal data regulations, such as GDPR in Europe, PDPA consists of various requirements in storing, governing and transmitting user's personal data.

2.2.2. PEPP-PT

Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT) is a technology protocol mainly for centralized contact tracing systems, where the App and personal data are controlled by the government health agency. It is a framework that adheres to GDPR and based on Bluetooth technology. The framework was released in April 2020 and soon adapted by many EU countries, such as Germany, France and Italy. It consists of four layers and three protocols, including data protection, secure communication and proximity measurement. [9]

2.2.3. DP-3T

Decentralized Privacy-Preserving Proximity Tracing (DP-3T) is a technology protocol mainly for decentralized contact tracing systems. Compared to PEPP-PT, DP-3T makes effort to largely reduce the privacy risk. It was released in April 2020 shortly after PEPP-PT. Different from PEPP-PT, the users' data are stored locally and only collected anonymously to a central database if the user gets infected.[10]

2.3. Decentralized System and Feature Embeddings

As mentioned in 2.1.2, in decentralized architecture, the software resides in each client's machine. In digital contact tracing, currently there is no completely decentralized app. TraceTogether is a partially decentralized solution, where the Bluetooth information is stored locally on the client's machine, while MoH maintains a central database of the infected user's data.

In TraceTogether, the user's data is encrypted. Therefore, the app protects users from data attack. However, MoH could see the data of infected users and their contacts. This requires people put trust into the MoH, which raised doubt in general public. The core question in maintain data privacy even against the government is that how to encrypt raw data while maintain the properties so that two sets of data could be compared. It naturally leads to a existing solution: feature embeddings.

Feature embedding is the process of transforming the data from original space into a new vector space, and extract data feature in the process [11]. There are various embedding algorithms on different types of data. For instance, Word2Vec for text embedding. Feature extraction of text or a sequence is challenging because of the arbitrary nature of data. There is a new sequence embedding algorithm called Sequence Graph Embedding (SGT), which is used as the embedding algorithm in this project.[7]

Suppose there is an GPS based, centralized mobile app which records down user's trajectory. Each place, such as a shopping mall or an MRT station, can be regarded as a node on the map. The trajectory in a specific time period T can be regarded as a sequence of nodes. And the vocabulary is the set of nodes. Using the SGT algorithm, sequence embeddings can be obtained in decentralized manner. The information sent to the central server is unreadable embeddings. This way, data privacy against the authority is ensured.

3. Equipment Resources and Costing

3.1. Hardware

The experiment of this PoC uses the following hardware:

MacBook Pro (13-inch, 2018, Four Thunderbolt 3 Ports)

Processor: 2.3 GHz Quad-Core Intel Core i5

Memory: 16 GB 2133 MHz LPDDR3

Graphics: Intel Iris Plus Graphics 655 1536 MB

Simulation, mock server and frontend all will be run on this laptop.

3.2. Software

3.2.1. React.js

React is a javascript library for building web frontend. It allows the developer to decompose a complex UI into smaller pieces of code called “component”. Each piece of UI can be written as a subclass of React.Component. A component takes in properties, which can be passed from higher level components, and returns a hierarchical view by the render method. The structure of components in this project will be showed in 3.2.3. With the components and state, a frontend built by react is highly responsive, reusable and has a clear structure. [12]

3.2.2. MapboxGL

MapboxGL JS is a javascipt library for displaying interactive maps on the frontend. In this project, it is used to display a map of Singapore.

3.2.3. Flask

Flask is a micro framework written in Python for developing backend. It does not have components like database abstraction layer and validations which are included in many other frameworks. However, it is flexible and supports many extensions, such as Flask-PyMongo used in this project. A simple mock server is needed in this project. Therefore, Flask is chosen as the framework for its simplicity. [13]

3.2.4. MongoDB

MongoDB is a open source document database that stores JSON like data. In this project, MongoDB Atlas is used for the simplicity of set up compare to local database. As stated in 3.2.1.3, Flask supports many extensions. Flask-PyMongo is a bridge between flask and PyMongo, which is a Python distribution for using MongoDB. [14]

3.2.5. Threading and concurrent.futures

In the implementation of the mobile simulation, threading is used for creating multi-agents. In Python threading, the threads are not actually executed in parallel. In this project, we only need a simulation where the events seem to happen at the same time, not speeding up the program. Thus, Python threading module is suitable. concurrent.futures is a module for managing threads in Python. It is a high-level interface compared to threading for asynchronous execution of callables. [15]

3.2.6. Shapely, Geopandas and pyproj

Shapely is a Python library for dealing with geometries. The Points are created and calculated using Shapely.geometry in this project. Geopandas is an open-source project for handling geospatial data. Pyproj is a Python interface to do coordinate transformations and cartographic projections. Earth has no precise geometric shape. There are plenty of coordinate reference systems, such as epsg:3414 for Singapore, epsg:3112 for Australia. Pyproj is used to transform the coordinates between different standards. [16]

3.3. Cost

The software used in this experiment are open source packages. The hardware are is a personal laptop. There is no additional cost incurred.

4. Project Schedule

FYP

NTU

Li Bingzi

Project Start: Mon, 1/6/2020

Display Week:

TASK	ASSIGNED TO	PROGRESS	START	END
Phase 1 Discussion and Design				
Reading and research		100%	1/6/20	4/7/20
First meeting and discussion		100%	4/7/20	8/7/20
System flow		100%	8/7/20	10/7/20
Usecase diagram, component diagram and techstack		100%	10/7/20	21/7/20
Phase 2 Frontend Implementation				
UI mockup, select framework		100%	10/7/20	14/7/20
Familiarise with React.js		100%	14/7/20	21/7/20
Implement frontend		100%	21/7/20	28/7/20
Testing and modification		100%	28/7/20	30/7/20
Phase 3 Embedding Module				
Research embedding techniques		100%	4/8/20	9/8/20
Discussion and testing		100%	10/8/20	17/8/20
Phase 4 Simulation				
Detailed Design		100%	12/8/20	23/8/20

Implementation	100%	24/8/20	2/11/20
Testing and modification	100%	3/11/20	18/11/20
Phase 4 Server			
Detailed Design	100%	4/8/20	11/8/20
Implementation	100%	18/11/20	17/1/21
Phase 6 Integration			
Server and simulation	100%	17/1/21	24/1/21
Frontend and server	100%	24/1/21	31/1/21
Testing and modification	100%	31/1/21	17/2/21
Phase 7 Documentation and Report			
Final Report	100%	17/2/21	22/3/21
Amended Final Report	0%	23/3/21	16/4/21
Presentation	0%	17/4/21	7/5/21
Github Documentation	10%	17/2/21	7/5/21

Table 1 Project Schedule

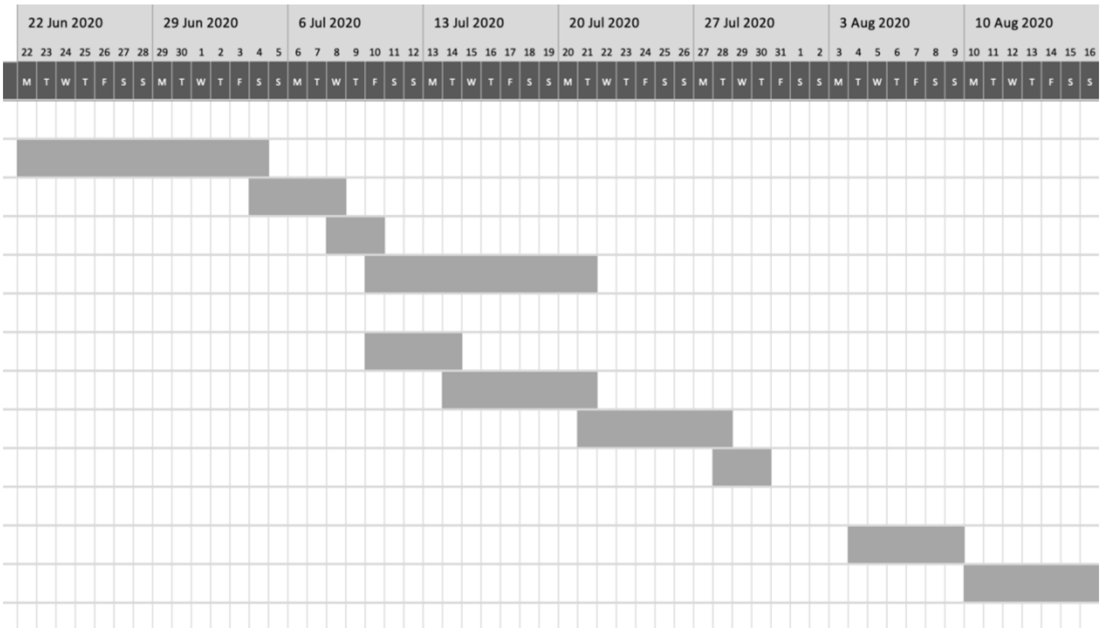


Figure 1 Part of the Gantt Chart

5. Methodology

5.1. Algorithm: SGT

The following arguments in this section are based on the studies of SGT, including the paper and python package documentations. Figure 2 are taken from the paper *Sequence Graph Transform (SGT): A Feature Embedding Function for Sequence Data Mining*. [7]

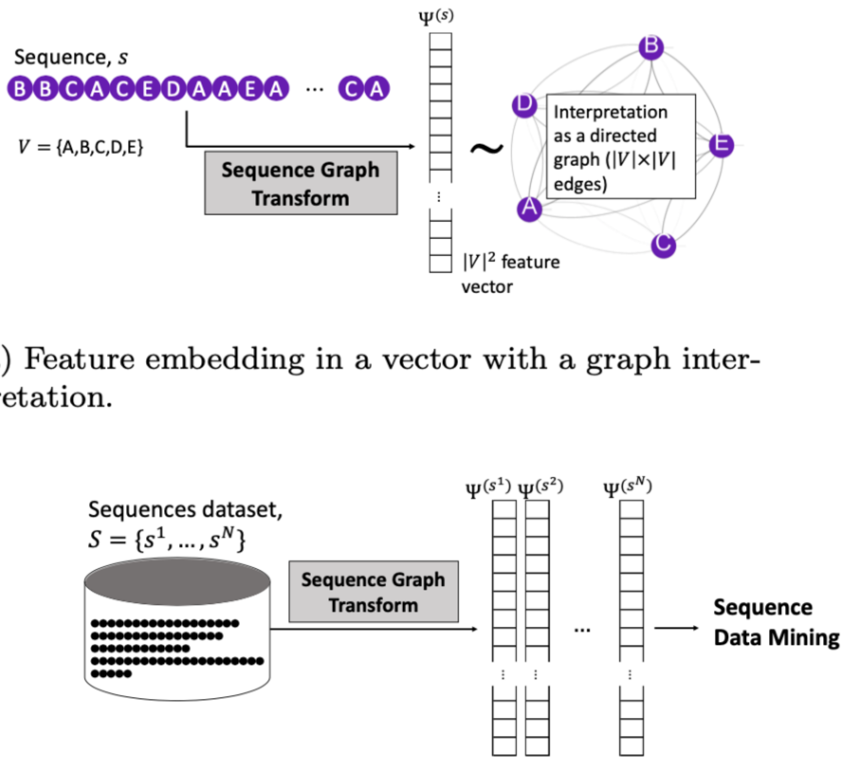
5.1.1. Intuition and Algorithm

As the name suggests, the intuition of Sequence Graph Transform comes from the structure of graph. The features are extracted from the relative position of alphabets in the sequence. The alphabet can be an event, in this project, a location. The reason of using relative position to represent the feature is that the similarity between two sequences is usually measured from the positional patterns.

The input of SGT is a sequence S where $s \in S \text{ and } s \in V$. The alphabet set is denoted as V. The sequence S is a feed-forward sequence. The quantitative measure of the effect on the relative position of the alphabets are denoted as $\phi(d(l,m))$ where l, m are the positions of two alphabets, and $d()$ is a distance measure.

Figure 2 shows a high-level overview of SGT. As shown in the figure, if interpreted as a graph, each alphabet is a node, and the feature is the edges between them. For example, in sequence $S = DAAB$, the relative position for pair $\{A, B\}$ is denoted as $\{(2,3),\ 4\}$. Similarly, the relative position between all alphabets can be extracted to get features in V^2 vector space.

There are two algorithms proposed in SGT. One is sequence embedding via sequence parsing. The other is sequence embedding via alphabets parsing. The sequence parsing method is faster for shorter sequences that length of the sequence $L < \sqrt{|V|}$. In our case, the length of sequence is the number of times per day that the device records the user's location. In the PoC, $L = 24$. $|V|$ is the total number of locations. In the PoC, locations are the MRT station in Singapore. However, when applied to real life, the number of locations will be much larger. The two algorithms are attached in the appendix.



(a) Feature embedding in a vector with a graph interpretation.

(b) Use of sequences' SGT embedding for data mining.

Figure 2 A High Level Overview of SGT

5.1.2. Advantages and Limitations

Sequence Graph Transform (SGT) is a novel method for sequence data mining. SGT is a algorithm that deals with both long term and short term dependencies without significant increase in computation. It transforms a sequence to a vector in finite dimensional feature space so that the result can be used in many tasks, such as similarity computation, Neural Network and so on.

The limitations are:

(1) The alphabet set cannot be small. For example, for DNA sequence, the alphabet set is {A ,C G, T}. The size is only 4. When applying the algorithm, the effectiveness will be diminished.

(2) The alphabets of the sequence need to be a single item. Eg. ABBCDEE is a valid sequence, <A><BB><CD><EE> is not.

The limitations of the algorithm do not affect this project.

5.1.3. Example of Using the Package

- parameters and methods related to the project:

alphabets	Alphabet set V, optional, will be computed by the algorithm if not input
kappa	Tuning the degree of long term dependency extraction, higher value means lesser long term dependency
flatten	Default true. If flatten, a vector is returned. Else a matrix is returned.
fit(sequence)	Input: one sequence Output: SGT embedding of the sequence
transform(corpus)	Input: new sequence which belongs to a previously fitted corpus Output: SGT embedding of the sequence
fit_transform(corpus)	Input: a list of sequences Output: SGT embedding of the corpus

Table 2 SGT class definition

- Example

```
In [1]: import sgt
        from sgt import SGT
        import numpy as np
        import pandas as pd

        sgt = SGT(flatten=True)
        sequence = np.array(["B","B","A","C","A","C","A","A","B","A"])
        sgt.fit(sequence)

Out[1]: (A, A)    0.090616
        (A, B)    0.131002
        (A, C)    0.261849
        (B, A)    0.086569
        (B, B)    0.123042
        (B, C)    0.052544
        (C, A)    0.137142
        (C, B)    0.028263
        (C, C)    0.135335
        dtype: float64
```

Figure 3 Basic Example of SGT

Suppose we have a corpus where each number is the id of the location and the sequence is the trajectory of the user. As seen in Figure 4, sequence 0 and sequence 1 have 3 common locations in consecutive manner. Sequence 0 and sequence 2 are similar in the beginning and end. Sequence 3 are different from the previous three.

```
In [7]: corpus = pd.DataFrame([[0, [1, 3, 4, 6, 10]],
                               [1, [1, 3, 4, 7, 8]],
                               [2, [1, 3, 5, 6, 10]],
                               [3, [2, 1, 8, 9, 7]],
                              ],
                              columns=['id', 'sequence'])
        # Learning the sgt embeddings as vector for
        # all sequences in a corpus.
        # mode: 'default'
        sgt = SGT(kappa=1,
                  flatten=True,
                  lengthsensitive=False,
                  mode='default')
        embedding = sgt.fit_transform(corpus)
```

Figure 4 Experiment with a Corpus

After getting the embedding, a simple calculation of Euclidean distance is done:

```
In [4]: embedding = np.array(embedding.iloc[:, 1:])

In [5]: n = len(embedding)
        distance = np.full((n, n), None)

        for i in range(n):
            for j in range(i + 1, n):
                distance[i][j] = np.linalg.norm(embedding[i]-embedding[j])

In [6]: pd.DataFrame(distance)
Out[6]:
```

	0	1	2	3
0	None	0.790689	0.783967	1.09689
1	None	None	0.965656	1.08845
2	None	None	None	1.09689
3	None	None	None	None

Figure 5 Calculating Euclidean Distance

As shown in figure 5, sequence 0 is indeed similar to sequence 1 and sequence 2. Comparing distance(0, 1) with distance(0, 2), it is clear that the SGT embedding has captured the long term relations between the alphabets. Comparing distance(0, 1) and distance(0, 2) with distance(0, 3), the difference is also captured by the algorithm.

In addition, other similarity computation method was experimented. The following is the result of applying cosine similarity on the same set of data.

```
pd.DataFrame(cos_sim)
```

	0	1	2	3
0	None	0.480378	0.489176	0
1	None	None	0.224966	0.0153208
2	None	None	None	0
3	None	None	None	None

Figure 6 Calculating Cosine Similarity

As shown by this matrix, cosine similarity reflects the same but better results in distinguish between similar and different sequences. The reason is that the embedding space is high-dimensional. The number of dimensions increases as the length of sequence increases. The detailed analysis is in Chapter 6.

5.2. Design Documents

5.2.1. System Overview

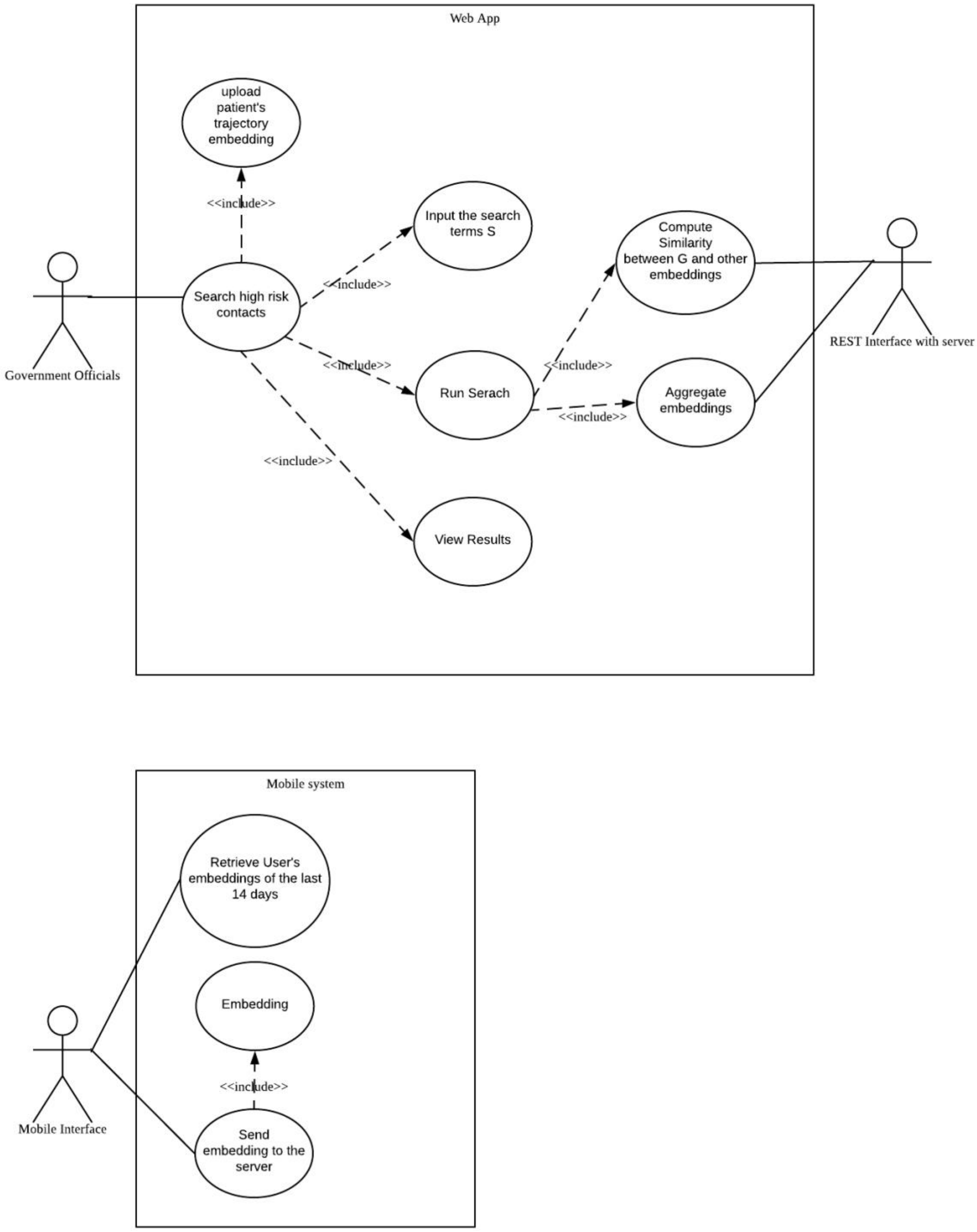


Figure 7 Use Case Diagram

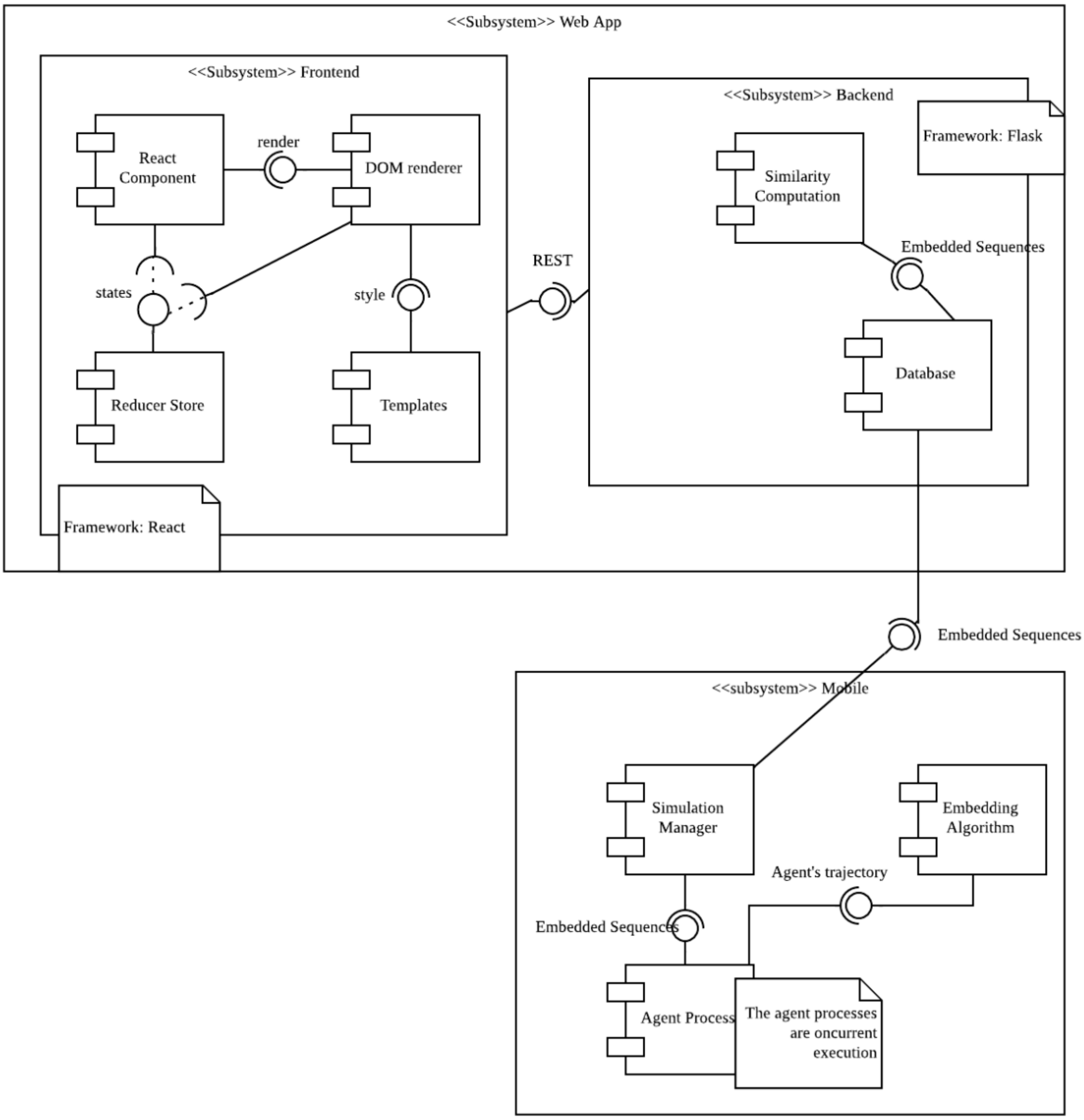


Figure 8 System Architecture Overview

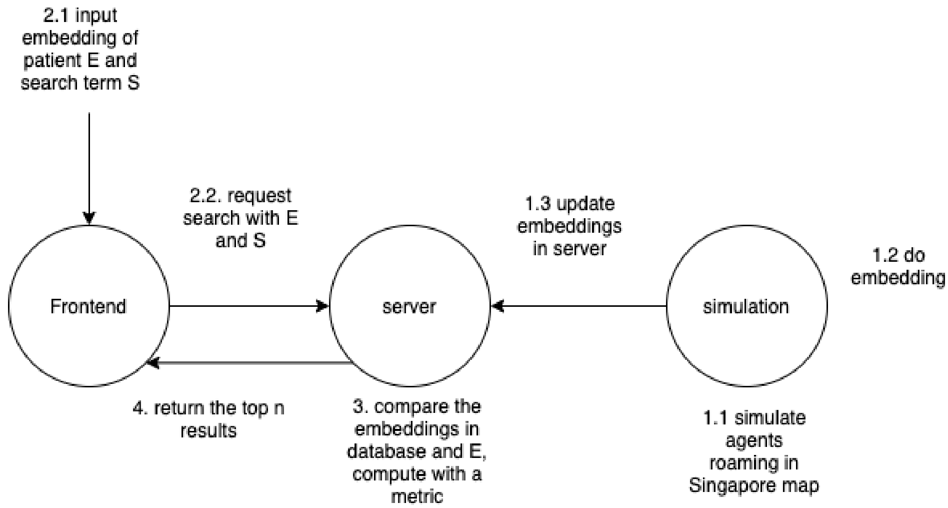


Figure 9 System Flow Overview

5.2.2. Frontend

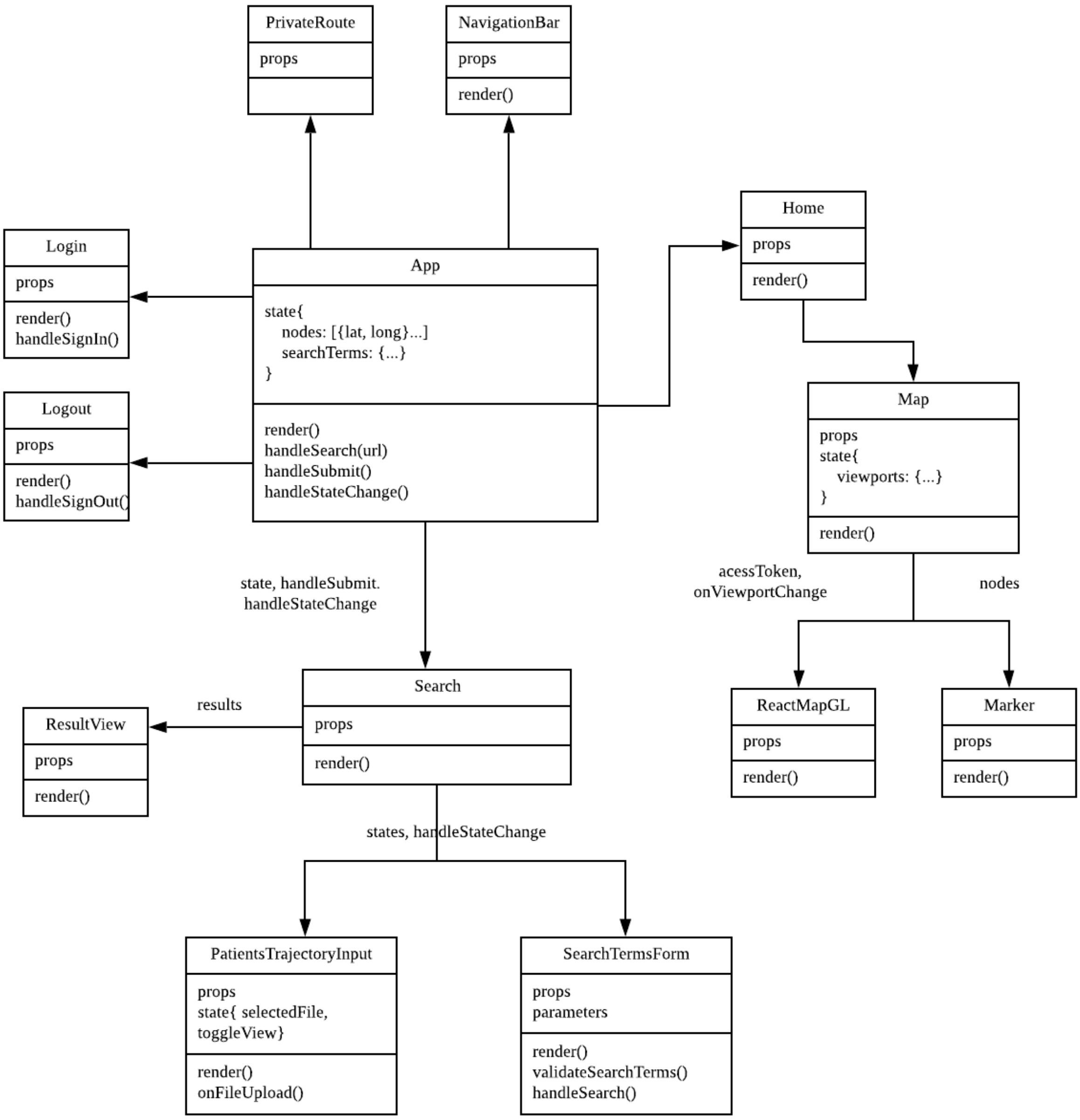


Figure 10 Detailed Frontend Design

5.2.3. Simulation

5.2.3.1. File Structure

```
.
├── main.py
├── mapfiles
│   └── SG4
│       ├── gadm36_SGP_0.cpg
│       ├── gadm36_SGP_0.dbf
│       ├── gadm36_SGP_0.prj
│       ├── gadm36_SGP_0.shp
│       ├── gadm36_SGP_0.shx
│       ├── gadm36_SGP_1.cpg
│       ├── gadm36_SGP_1.dbf
│       ├── gadm36_SGP_1.prj
│       ├── gadm36_SGP_1.shp
│       ├── gadm36_SGP_1.shx
│       └── license.txt
├── node_data
│   └── mrt_lrt_data.json
└── simulator
    ├── Agent.py
    ├── Env.py
    ├── Map.py
    ├── __init__.py
    ├── __pycache__
    │   ├── __init__.cpython-37.pyc
    │   ├── agent.cpython-37.pyc
    │   ├── constants.cpython-37.pyc
    │   ├── env.cpython-37.pyc
    │   └── map.cpython-37.pyc
    └── constants.py
```

Figure 11 File Structure in Mobile Simulation

File Name	Explanation
main.py	The starting point of the simulation, create and run multiple threads of agents
mapfiles/*	The folder containing maps of Singapore
node_data/*	A json file of geospatial data of MRT stations in Singapore. This is the mock data used in PoC. In real life applications, this would be SafeEntry checkpoints in Singapore.
simulator/*	The package containing all classes for the building simulation
Agent.py	Simulate an agent (with mobile app) randomly roaming in Singapore

	boundary. The program records down the trajectory, do embeddings and send them to the server.
Map.py	Containing methods of handling the mapfiles
Env.py	Create an environment for the agent
constants.py	containing some constants while running the simulation (modifiable for the next round of simulation)

Table 3 Explanation of Files

5.2.3.2. Simulation Parameters

constants	usage
FILE_LOC, NODE_DATA_LOC, SERVER_URL	the location of mapfiles and node data server url to make http request, in the PoC we use localhost
SG_CRS	set to 3414 (epsg:3414): the spatial reference system for Singapore unit: meter
UPDATE_PERIOD	the time period for the agent to update its location
SIMU_TIMESCALE	the timescale of simulation time/real time
DEBUG	debug mode that will only instantiate one agent and output some logs
SIMU_DAYS	simulation time length unit: s
min_speed, max_speed	will be used in generating random speed of the agent

	by default, min_speed = 0, max_speed = 14 (driving and MRT) unit: m/s
record_range	The GPS coordinate might not be exactly on the nodes. The simulation records coordinate (long, lat) as on node N if (long, lat) is within record_range of node N In the experiment part, this parameter will be tuned.

Table 4 Simulation Parameters

5.2.3.3. Algorithm

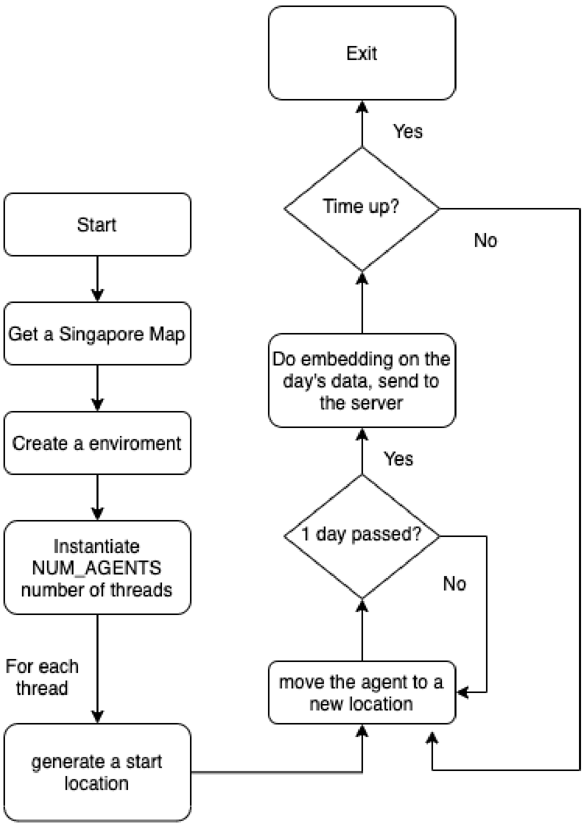


Figure 12 Simulation Flowchart

```
get (long_min, long_max, lat_min, lat_max) bounds of the Singapore map
while True:
    generate a random location (long, lat) within bounds
    if (long, lat) lies within the actual Polygon of Singapore map:
        return location
```

Figure 13 Pseudocode for Random Location Generation

```
the agent has 50% chance to move or stay at current location
if agent moves:
    generate a direction d (radian)
    generate a random speed s in range(min_speed, max_speed)
    calculate new location
    while the location not in Singapore map:
        center = (long_min + long_max)/2, (lat_min + lat_max)/2
        generate a random speed s in range(min_speed, max_speed)
        move the location towards center at s

    assign new location to agent
```

Figure 14 Pseudocode for Updating Agent's Location

5.2.4. Server

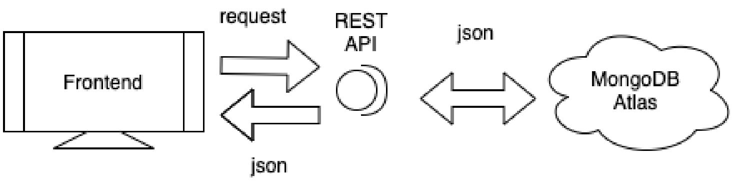


Figure 15 Server Overview

endpoint	method	function
/upload	POST	receive a file (json) from request, save it to the database
/users/validate	POST	validate a user's information
/users/signin	POST	business logic of sign in

/users/signout	POST	business logic of sign out
/embedding	POST	called by the simulation save the embedding sent by simulation into the database
/search	GET POST	retrieve other user's embeddings from the database, compute the similarity with patient's vector, then return the top n most similar user id.

Table 5 Server Endpoints

6. Experiment Result and Discussion

6.1. Mobile Simulation

6.1.1. Simulate a Trajectory

The simulation of one agent is done with the following parameters:

SIMU_DAYS = 14

DEBUG = True

UPDATE_PERIOD = 3600 # unit: s, meaning update the server every hour

SIMU_TIMESCALE = 36000 # simulation time / real time

record_range = 2000 # unit: meter

```
[[ 83  26  83  83  83  83  83  83  83  83  -1  34  34 136  85  85  53 116
  116  -1  -1 139 139  86]
 [108  83  13  13  13  85  -1  44  55  55 137  35   9   9   9   9   9  93
   93  93  93  93  23  23]
 [ 23  77  64  84  84  -1  -1 138  -1  -1  -1  37  -1  83  83  83  -1  -1
   -1  -1  -1  -1  -1  -1]
 [ -1  54  54 139 139 139 139  -1  -1  -1  51  83  83  11  10  72  72  72
   72  72  -1  57  11  11]
 [133 133 133 133 133  13  13  13  13  68  66  66  66  13  90  -1  -1  -1
   35  56  54  54  54  54]
 [113  87 100 100  35  35  35   2   2   2  -1  -1  -1   9   9   9  12  12
   12   9   9   9   9  -1]
 [ 85  10  10  10  86 102  61  48  13  13 139 139  23   4  86  85  85  85
   85  62  -1  -1  35  35]
 [ 13  54  13  57  -1  19 118 118  39  92  92   4   4   4   4 140 140 140
   19  -1  -1  -1  -1  83]
 [ 83  14  82  88 107 101  83 126 126 100  -1  39  84  84  -1  -1  -1 101
  101 101 101  -1  -1  59]
 [ 59 106  83  83  -1  -1  -1  -1  -1 122 122 122  -1  -1  -1  -1  -1  -1
   -1  -1  -1  -1  -1  -1]
 [ -1  -1  -1  82 100 101 101 101 101 101  99  30  30  -1  -1  -1 101 101
  101  11  82  -1 138 138]
 [138 138  13  -1  -1  -1  36  98  -1  -1  -1  -1  -1  -1  -1  -1  -1  12
   12  -1  56  56  99  -1]
 [ -1  35  35  -1  -1  -1  -1  18  18  83  -1  22  22  22  22 102 102  86
   99 116 116 116 116 113]
 [113  99  13  85  85  -1  -1  -1  86  86   7   7  84  84  84  84  84  84
   -1  -1   2  33  33  10]]
```

Figure 16 A Random Trajectory Array

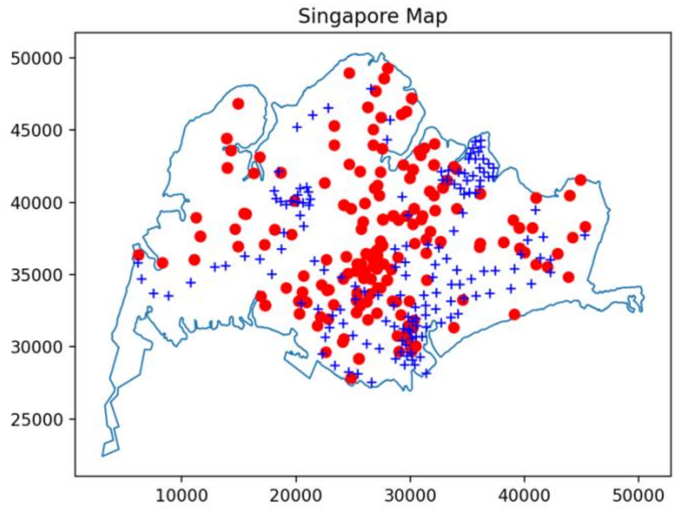


Figure 17 Trajectory Map

The number in the array is the id of each node in the map. In the map, the MRT stations are marked as blue plus signs. The red dot is the places the agent has been to. The '-1' means that the coordinate of the agent is not near any node.

As shown by the map, the simulation system is a rather random simulation aims to generate the trajectory data rather than simulating mobile users in real life. In real life cases, the user is unlikely to go out at midnight, will only go to a few places on working days. Different groups of users have different behavior patterns etc. These can be considered as possible future improvements of the simulation.

Another problem is that there are some coordinates that are not near any node. In the real application, this is unlikely to happen. In a real application such as SafeEntry, the nodes cover every shop and residential buildings. Large number of undefined nodes will also affect the accuracy of embeddings. There are two solutions. One is to introduce more nodes (such as bus stops, shopping malls) into the simulation. Another one is to tune the record_range parameter so that there are less undefined locations.

6.1.2. Tuning Parameters

record_range (m)	number of undefined locations out of 14x24 = 336 recorded points
500	294
1000	212
2000	88
3000	36
4000	35

Table 6 Tuning record_range

The record_range in the simulation is set to 3000.
This is for simulation in the PoC only. In real life scenario, the number of record locations are much larger. The recording is done by GPS. The number of undefined locations in a user's records will be low.

6.1.3. Embedding

Embeddings are done once a day in simulation time. Locations are recorded once an hour in simulation time. In total, every embedding is done on array of length 24. Every user has an array of 14 embeddings kept in the database. The record frequency can be further changed, for example every 30 minutes, depends on the real-life scenario.

The following steps show how an individual agent does the embedding in a jupyter notebook. The exact same method is applied to the simulation. The two sequences are taken from user trajectories randomly generated by the simulation.

```
pd.options.display.max_colwidth = 200
corpus
```

	id	sequence
0	0	[20, 57, 100, 116, 116, 116, 3, 3, 128, 67, 67, 67, 135, 135, 88, -1, -1, -1, -1, 96, 96, 78, 119, 83]
1	1	[82, 82, 82, 82, 12, 40, 50, 86, 86, 31, 1, 45, 56, 56, 56, 35, 35, 12, 50, 50, 33, 33, 33, 117]
2	2	[20, 57, 100, 116, 116, 116, 3, 86, 86, 31, 1, 45, 56, 56, 56, 35, 35, 12, 50, 50, 33, 33, 33, 117]

```
embedding
```

	id	(-1, -1)	(-1, 0)	(-1, 1)	(-1, 2)	(-1, 3)	(-1, 4)	(-1, 5)	(-1, 6)	(-1, 7)	...	(156, 147)	(156, 148)	(156, 149)	(156, 150)	(156, 151)	(156, 152)	(156, 153)	(156, 154)	(156, 155)	(156, 156)
0	0.0	0.237349	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

3 rows × 24965 columns

Figure 18 Embedding of two Trajectories

The embeddings show the limitations of the solution. When we have 157 locations, the length of one embedding is 24964 (excluding the id column).

For the number of locations n,

$$L(embedding) = (n+1)^2$$

As the number of locations increases, the length of embeddings increases exponentially. In addition, the vector or the matrix before flattening is very sparse. To save the storage, several techniques can be used to compress the vectors (see future improvement).

6.1.4. Similarity Measures

In Chapter 5, two different similarity computation method are applied on short sequences. In the following step, we further compare them on the same set of data as figure 18.

	0 and 1	0 and 2	1 and 2
Euclidean distance	1.68	1.54	0.94
cosine similarity	0.00	0.27	0.71

Table 7 Similarity Measures

As shown by Table 7, although both measures reflect the similarity of the original data, Euclidean distances are close to each other in the vector space. The reason is that the average and maximum Euclidean distance of vectors become closer as the number of dimensions grows. This is the curse of dimensionality.

In comparison, cosine similarity better reflects the features. The curse of dimensionality applies to cosine similarity as well. However, cosine similarity works better in this case because of the context. Euclidean distance is commonly applied to dense data, while cosine similarity is commonly used in domains like text processing where the data or embeddings are very sparse.

6.2. Server and Frontend: Use Cases

6.2.1. Homepage (entry point)

Entering the homepage, there’s an interactive map which can be used to display covid19 infected area in future implementations. There’s a menu bar, and a sign in option. Click on “Search” in the menu bar or “Start search” button without being logged in will bring the user to the login page.

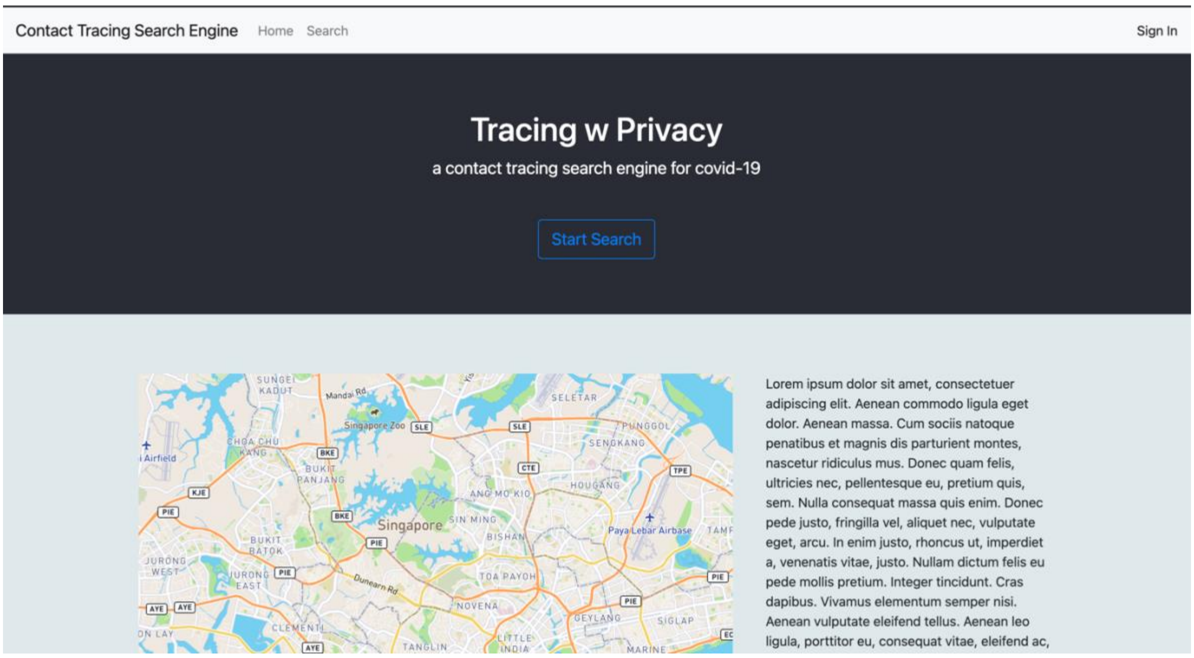


Figure 19 Homepage UI

6.2.2. Login page

The frontend web page has responsive web design. This Sign In page is screenshotted with 50% web page width.

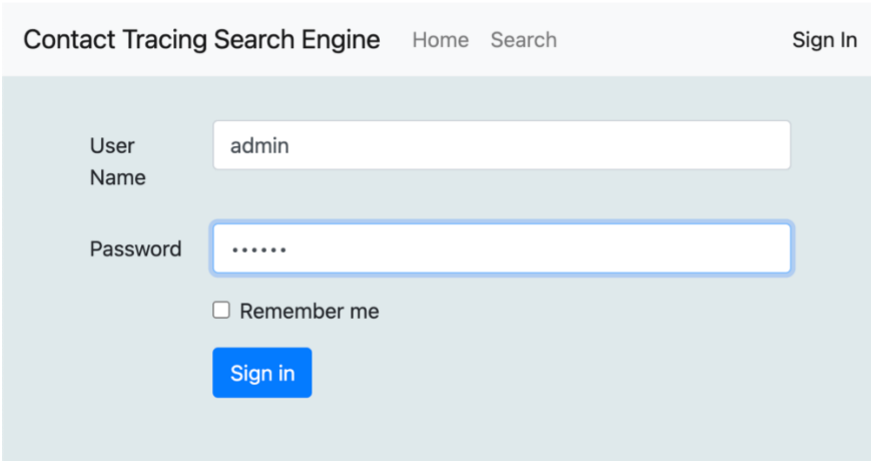


Figure 20 Sign In Page

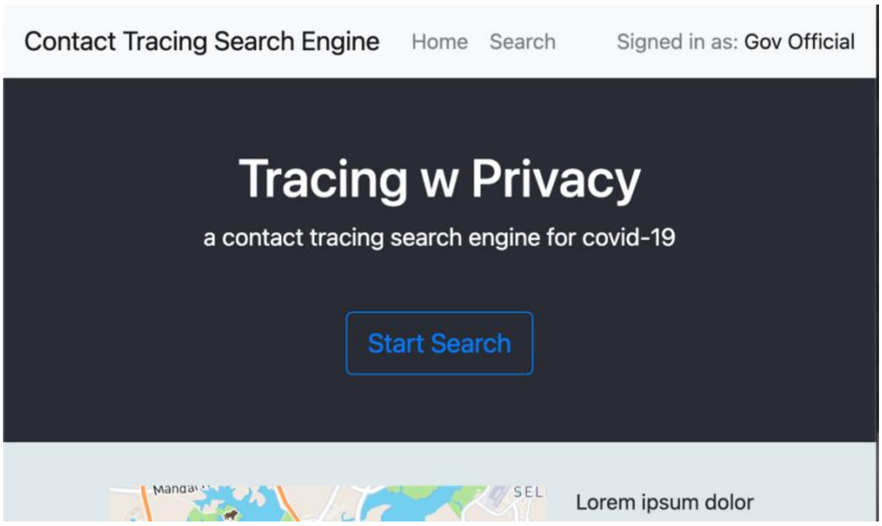


Figure 21 After Sign In

6.2.3. Search

To search the users with similar trajectories with the patient, the government official needs to upload the patient’s trajectory embedding. It is assumed they can retrieve patient’s embedding from the mobile app.

Then the government official needs to key in the search terms. The search date range can be omitted, the system will compute according to the past 14 days data if omitted. The field “entry to display” is the number of search result to return.

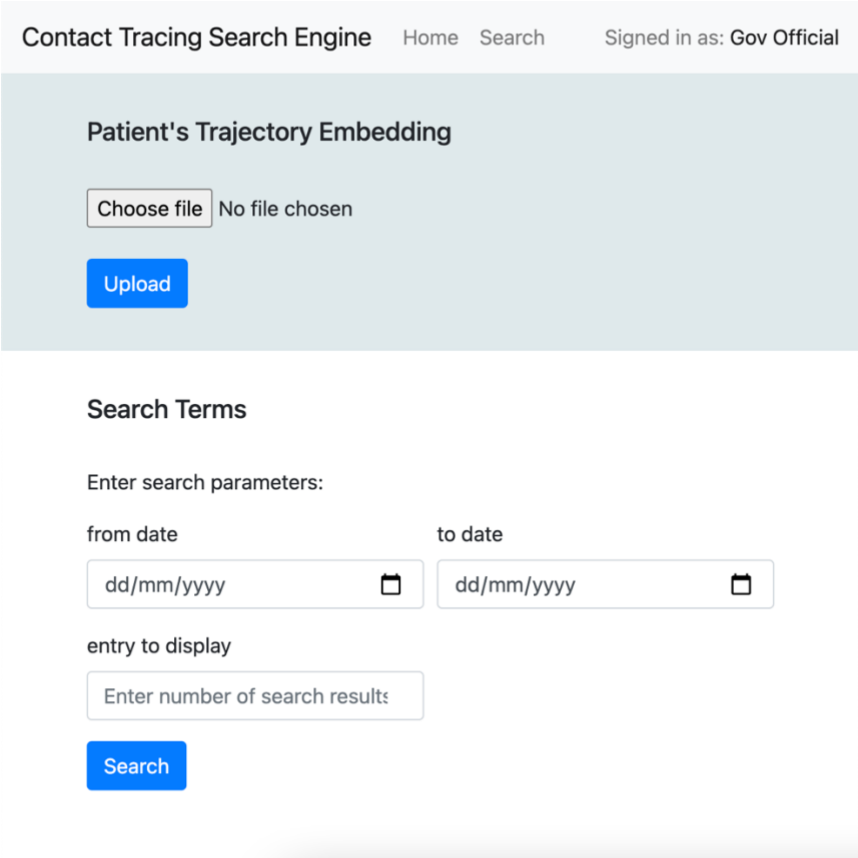


Figure 22 Search Page

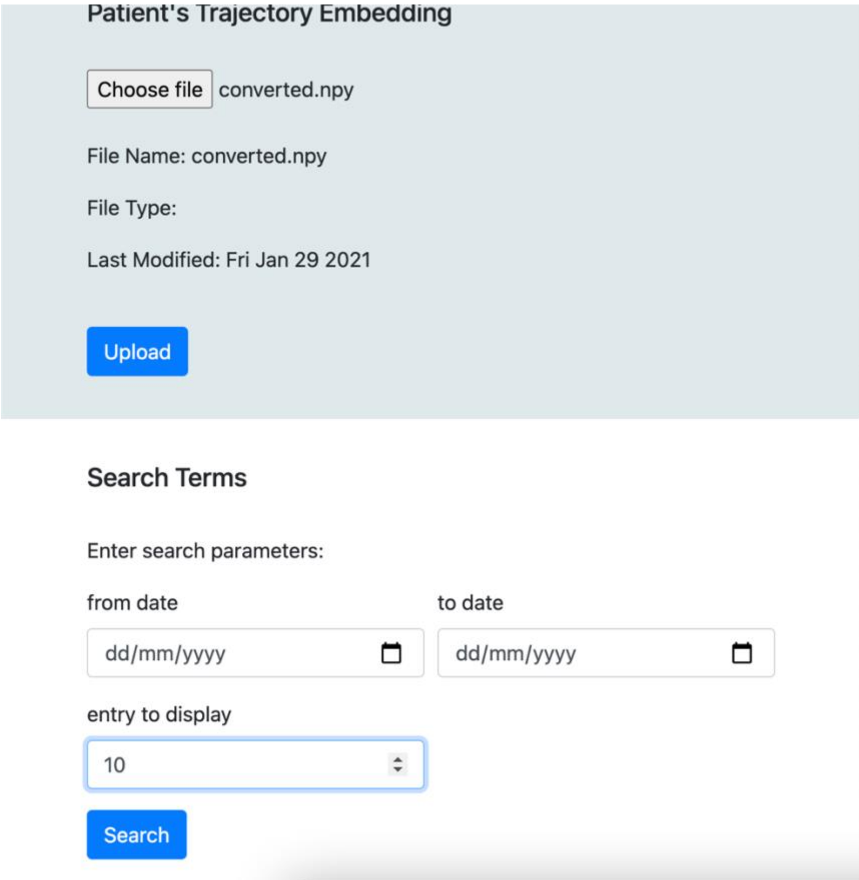


Figure 23 Uploading and Filling In Search Terms

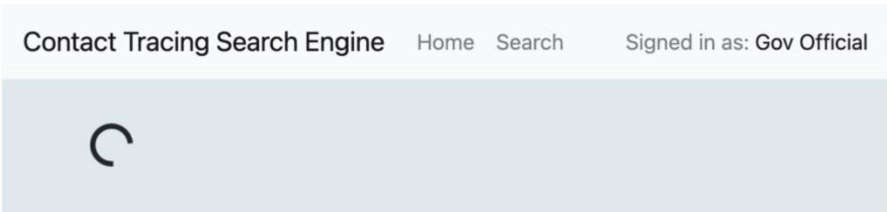
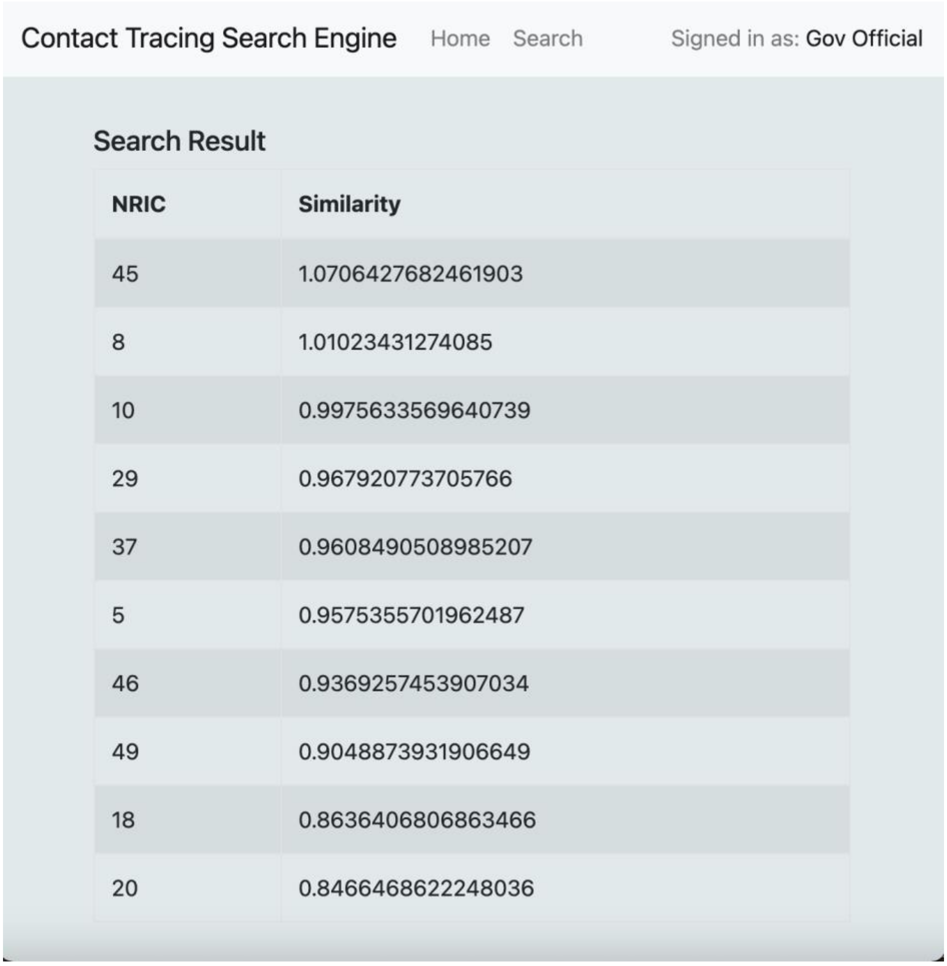


Figure 24 Loading Page while Searching



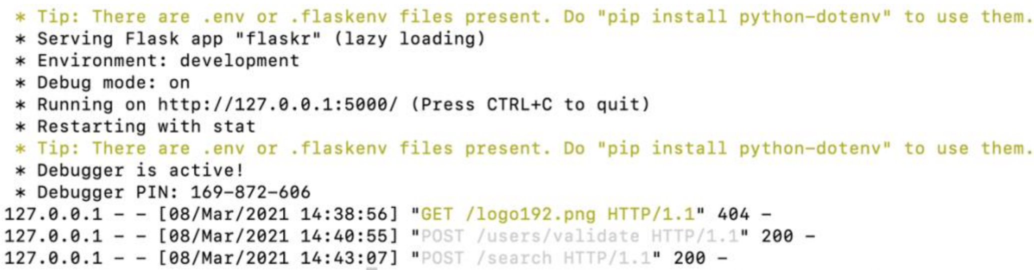
Contact Tracing Search Engine Home Search Signed in as: Gov Official

Search Result

NRIC	Similarity
45	1.0706427682461903
8	1.01023431274085
10	0.9975633569640739
29	0.967920773705766
37	0.9608490508985207
5	0.9575355701962487
46	0.9369257453907034
49	0.9048873931906649
18	0.8636406806863466
20	0.8466468622248036

Figure 25 Search Results Displayed

The similarity displayed here is computed by simply adding the similarity of each day's trajectory. In real-life case, more complex way of calculating the similarity score is needed with the advice from infectious disease scientists.



```
* Tip: There are .env or .flaskenv files present. Do "pip install python-dotenv" to use them.
* Serving Flask app "flaskr" (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Tip: There are .env or .flaskenv files present. Do "pip install python-dotenv" to use them.
* Debugger is active!
* Debugger PIN: 169-872-606
127.0.0.1 - - [08/Mar/2021 14:38:56] "GET /logo192.png HTTP/1.1" 404 -
127.0.0.1 - - [08/Mar/2021 14:40:55] "POST /users/validate HTTP/1.1" 200 -
127.0.0.1 - - [08/Mar/2021 14:43:07] "POST /search HTTP/1.1" 200 -
```

Figure 26 Response by the Server

```
0.12101973700694162
0.005142634506365144
0.0
0.07785365676098556
0.029443080458172014
0.0332615200815407
0.008787948513589685
0.015616411992648457
0.023108037352935134
0.011413275613709922
0.23185920698131077
0.05701623155421706
0.007098006070476532
0.02298743443271611
0.20600103785095236
0.09041463021763395
0.0
0.0665263058191502
0.05161677094532352
0.01447442409981783
0.20059647899853789
0.18692952147763375
0.0001404072857588379
0.009849964511523953
0.1804825217525028
0.032923358612263205
0.011743697183892409
0.013806012885938199
0.11522779011432706
0.03971708236810471
0.0
0.0858872309132354
0.027578913889999532
4.413196899711614e-06
```

Figure 27 Partial Logs of Cosine Similarity between Embeddings

7. Conclusion and Future Improvements

7.1. Conclusion

To conclude, a PoC of decentralized contact tracing system is developed through this project. The PoC shows that it is possible to use sequence embedding techniques to perform contact tracing with low risk of privacy intrusion. To accomplish this goal, a simulation of moving agents within Singapore map, a server that handles the similarity computing and the frontend for searching were implemented. The results imply that embedding based decentralized contact tracing is a potentially effective solution as long as scalability issues are solved when applying the solution to a whole Singapore population.

7.2. Limitations and Future Recommendations

7.2.1. Scalability

Due to the limited computational resources of a personal laptop, the number of agents simulated by the PoC is far less than the number of actual users when a similar app is put into use. In addition, the alphabets in the sequence, in other words the record locations in Singapore, is MRT stations. In total, it is 157 points. For a real app, the length of alphabets will increase dramatically, likely 1000~10000 times more. As shown in 4.1.3, the length of vectors grows cubically as the length of alphabets increases. What is more, the vector which is flattened from a matrix is very sparse. Storing these sparse vectors are a huge waste of database storage.

To store a sparse matrix, there are several techniques available:
(1) DOK (Dictionary of Keys): Storing the matrix as a dictionary with key (row, column) and value (non-zero value). The length of matrix (m, n) is known. The (row, column) not in the dictionary is zero.
(2) LIL (List of Lists): Different from DOK, it has one list per row and stores the non-zero value in the format of (column, value) as a node in the list. The list per row is sorted by column number.
(3) CSR (Compressed Sparse Row): CSR represents a sparse matrix in three arrays, including non-zero values, rows and columns index. Compared to the methods above, CSR allows quick access to rows and matrix-vector multiplications.

Besides storage, search time is also an issue. With 50~100 users, the search time is 2~3 seconds. The algorithm computes the patient's embeddings against each stored embedding of other users and rank them, which means O(nlgn) complexity for n users. As the number of users increase to a portion of Singapore's population, responsive search will not be feasible even with better computational resources.

One of possible solution to speed up the search is clustering. The centroids of each cluster are kept in a list. When searching, the similarity between centroids and the patient are computed first. Then, N nearest clusters will be searched. However, this solution does not guarantee a correct answer, which is a huge risk in life-death situation like covid-19 tracing. Another solution is parallelism. It requires large computational resources but guarantees accuracy.

7.2.2. Security

The process after embedding is secure for the user's data, since transmission and calculation are all done using embeddings. If a similar system is put into use, the developers still need to pay attention to the access control.

7.2.3. Usability

In the PoC, the mobile apps are simulated. The embedding of the patient is generated by the simulation as a .npy file. The usability of both mobile apps and the website needs to be considered if a similar contact tracing system is implemented in real life. The way to get a patient's embeddings needs to be simple. The process of sending out notifications should be efficient. Besides, the elderly and the disabled needs to be considered when building a real tracing system.

Reference List

[1]"WHO Coronavirus Disease (COVID-19) Dashboard", *Covid19.who.int*, 2021. [Online]. Available: https://covid19.who.int/. [Accessed: 19- Feb- 2021].

[2]J. LI and X. Guo, "Global Deployment Mappings and Challenges of Contact-tracing Apps for COVID-19", *SSRN Electronic Journal*, 2020. Available: 10.2139/ssrn.3609516.

[3]Q. Tang, "Privacy-Preserving Contact Tracing: current solutions and open questions", p. 9. Available: https://arxiv.org/pdf/2004.06818.pdf. [Accessed 19 February 2021].

[4]"Does the Government have access to the data? What will the data be used for?", *Team SafeEntry*, 2021. [Online]. Available: https://support.safeentry.gov.sg/hc/en-us/articles/900000700203-Does-the-Government-have-access-to-the-data-What-will-the-data-be-used-for-. [Accessed: 19- Feb- 2021].

[5]"Singapore reveals Covid privacy data available to police", *BBC News*, 2021. [Online]. Available: https://www.bbc.com/news/world-asia-55541001. [Accessed: 19- Feb- 2021].

[6] Kumar, P., Krishna, P.R., Raju, S.B.: Pattern discovery using sequence data mining:
applications and studies. Information Science Reference (2012)

[7]C. Ranjan, S. Ebrahimi and K. Paynabar, "Sequence Graph Transform (SGT): A Feature Embedding Function for Sequence Data Mining." Available: https://arxiv.org/pdf/1608.03533.pdf. [Accessed 19 February 2021].

[8] D. Schuff and R. St. Louis, "Centralization vs. decentralization of application software", Communications of the ACM, vol. 44, no. 6, pp. 88-94, 2001. Available: 10.1145/376134.376177.

[9]"pepp-pt/pepp-pt-documentation", GitHub, 2021. [Online]. Available: https://github.com/pepp-pt/pepp-pt-documentation. [Accessed: 22- Feb- 2021].

[10]"DP-3T/documents", *GitHub*, 2021. [Online]. Available: https://github.com/DP-3T/documents. [Accessed: 22- Feb- 2021].

[11]E. Golinko and X. Zhu, "Generalized Feature Embedding for Supervised, Unsupervised, and Online Learning Tasks", Information Systems Frontiers, vol. 21, no. 1, pp. 125-142, 2018. Available: 10.1007/s10796-018-9850-y.
[12]"Getting Started – React", *Reactjs.org*, 2021. [Online]. Available: https://reactjs.org/docs/getting-started.html. [Accessed: 16- Mar- 2021].
[13]"Welcome to Flask — Flask Documentation (1.1.x)", Flask.palletsprojects.com, 2021. [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/. [Accessed: 16- Mar- 2021].
[14]"Flask-PyMongo — Flask-PyMongo 2.3.0 documentation", Flask-pymongo.readthedocs.io, 2021. [Online]. Available: https://flask-pymongo.readthedocs.io/en/latest/. [Accessed: 16- Mar- 2021].
[15]"concurrent.futures — Launching parallel tasks — Python 3.9.2 documentation", Docs.python.org, 2021. [Online]. Available: https://docs.python.org/3/library/concurrent.futures.html. [Accessed: 16- Mar- 2021].
[16]K. (https://www.klokantech.com/), "EPSG.io: Coordinate Systems Worldwide", Epsg.io, 2021. [Online]. Available: https://epsg.io/. [Accessed: 16- Mar- 2021].