INTERNATIONAL TELECOMMUNICATION UNION **TELECOMMUNICATION STANDARDIZATION SECTOR** STUDY PERIOD 2022-2024		**FOCUS GROUP ON AUTONOMOUS NETWORKS (FG-AN)**
		AN-I-296
		Original: English
Question(s):	NA	February 2023 (TBC)
INPUT DOCUMENT		
Source:	Indian Institute of Technology, Bhilai	
Title:	Report on activities for Build-a-thon 2022 from AI_ML_SSD team	
Contact:	Debanka Giri, Project Scientist, IIT Delhi	E-mail: debanka.giri@cse.iitd.ac.in
Contact:	Sudev Kumar Padhi, Ph.D., IIT Bhilai, India	E-mail: sudevp@iitbhilai.ac.in
Contact:	Mohit Kumar, B.Tech, IIT Bhilai, India	E-mail: mohitkumar@iitbhilai.ac.in
Contact:	Dr. Sk Subidh Ali Mentor	E-mail: subidh@iitbhilai.ac.in

Keywords:	Deep learning, Long Short Term Memory, Deep neural Network
Abstract:	This contribution provides a report on activities by the AI_ML_SSD team towards the Build-a-thon 2022. We analyze the Indian Institute of Technology Delhi use cases, "Slip Detection (and Force Estimation)" and "Object Detection" in a robotic arm, and produce a design as per the reference design in the Build-a-thon repository. We also provide the corresponding code based on the reference code in the Build-a-thon 2022 repository. After analyzing the use cases, we trained two ML models, one for "Slip Detection (and Force Estimation)" and another one for "Object Detection." We have tested and validated the models for the use cases. A demo video of the simulation of the robotic arm picking up the object is also given for reference, and how the use cases "Slip Detection (and Force Estimation)" and "Object Detection" are estimated by our ML models.

Scope

This document provides a report on the analysis of the Indian Institute of Technology Delhi use cases "Slip Detection (and Force Estimation)" and "Object Detection." The report includes the following:

- Analysis of the use case with examples
- A design of the use case
- Code to produce the graph-based design based on neo4j per the reference code provided in the Build-a-thon repo.
- Demo Video (with caption)
- Screenshots of Models and simulator
- ML Model architectures used
- Accuracy and hyperparameters of the Models

1 References

[FGAN-use cases] ITU-T Focus Group Autonomous Networks Technical Specification "Use cases for Autonomous Networks" https://www.itu.int/en/ITU-T/focusgroups/an/Documents/Use-case-AN.pdf

[Build-a-thon 2022] https://github.com/vrra/FGAN-Build-a-thon-2022

[FG AN Arch framework] Architecture framework for Autonomous Networks, https://www.itu.int/en/ITU-T/focusgroups/an/Documents/Architecture-AN.pdf

[Demo Videos] https://drive.google.com/drive/folders/1Vih2QWYcVxqYS6VP-sjBcO-28p7mEaP8?usp=sharing

[Code] https://github.com/sudev007/Build_a_THON

[TabNet] TabNet: Attentive Interpretable Tabular Learning

[IITD use case] https://bhartischool.iitd.ac.in/build_a_thon/index.html

2 Abbreviations and acronyms

This document uses the following abbreviations and acronyms:

FC Fully-Connected

BN Batch Normalisation

GLU Gated Linear Unit

LSTM Long Short Term Memory

DNN Deep neural Network

ML Machine Learning

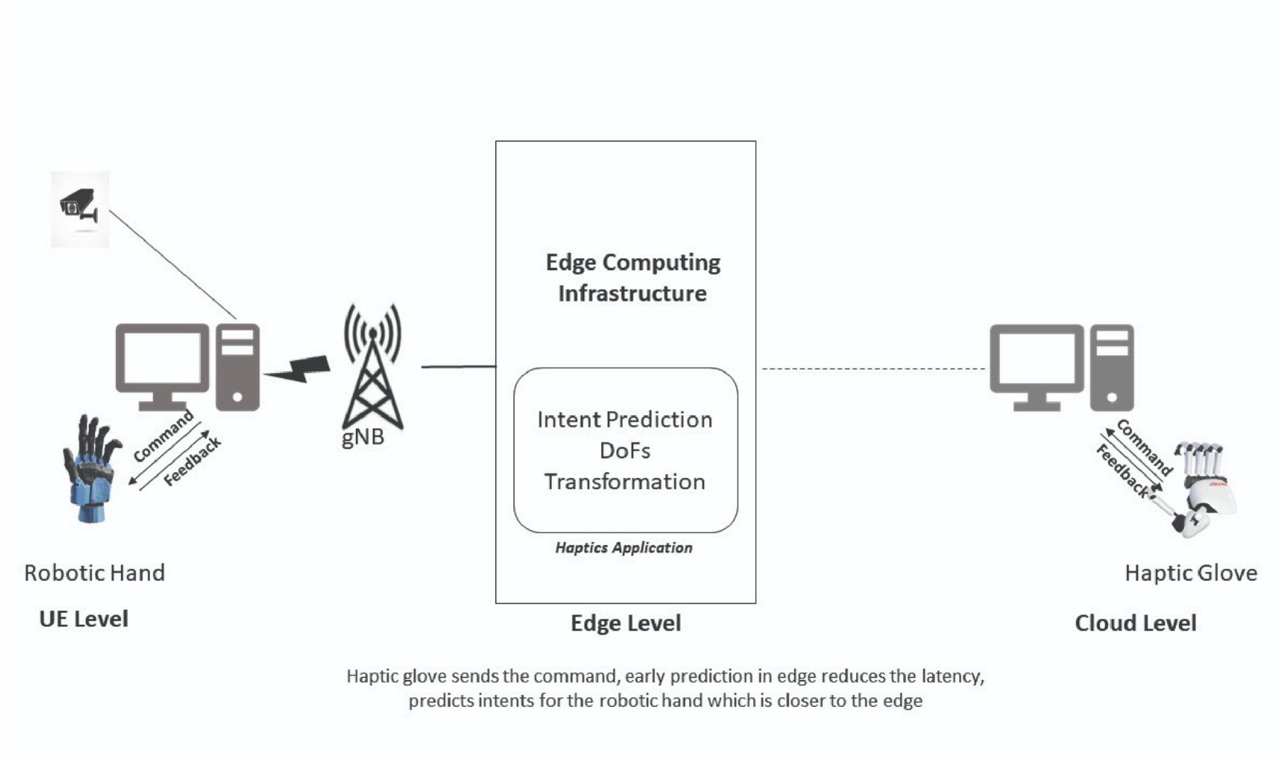
MEC Multi Edge Computing

3 Conventions

None

4 Introduction

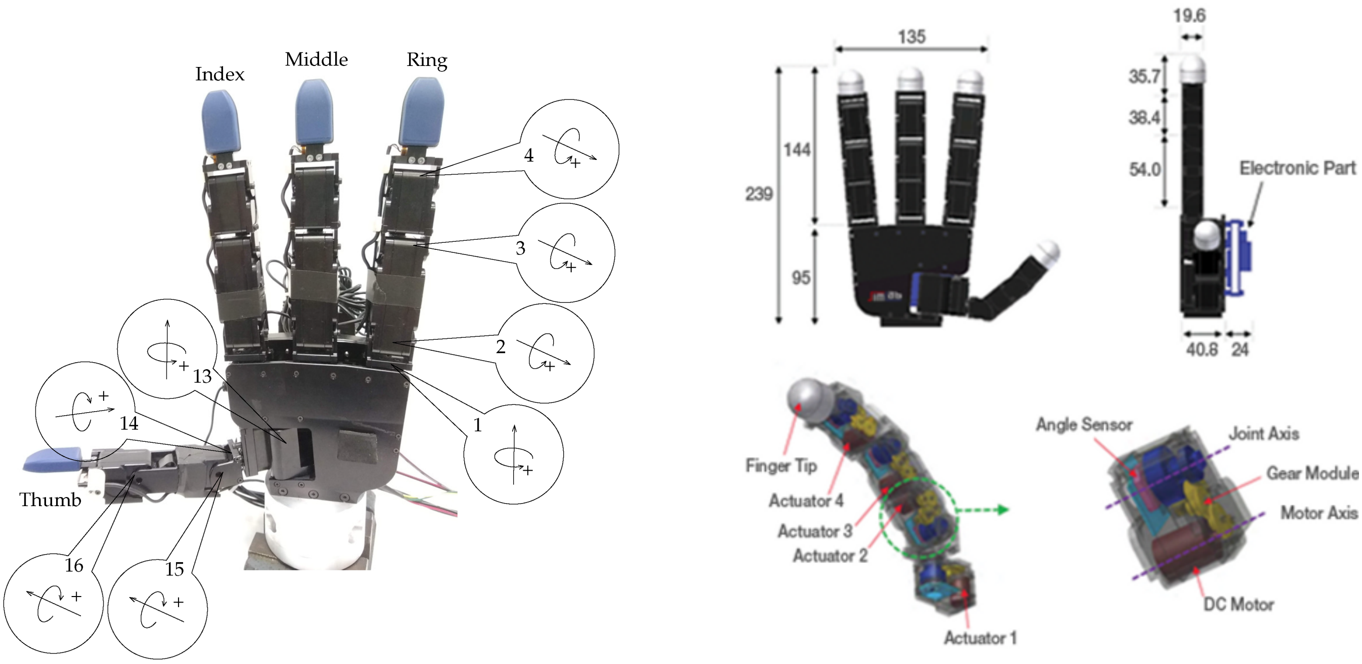
The Indian Institute of Technology Delhi has developed a MEC Test Bed integrated with 5G Core conforming to ETSI standards . One of the things made on the test bed was an Allegro Hand, a robotic hand that can do things like pick up an object, hold it down, grasp it, paint, etc. To do the tasks listed above, you need an operator with a haptic glove that controls how the allegro hand moves. An allegro hand can be used to work remotely, like changing cables. At the same time, the operator is the cable guy, or even a critical task, like an operation, while the operator is a doctor. So, various tasks need different operators to perform the tasks using an allegro hand. With the development of technology, especially machine learning, a machine learning model can take the place of an operator and do all the tasks that need to be done. An operator has to do several smaller tasks to finish a given task, such as changing a cable, doing an operation, etc. The Indian Institute of Technology, Delhi, has found two subtasks that are often used for different applications. For each of these tasks, we made a machine-learning model.



[ref:https://bhartischool.iitd.ac.in/build_a_thon/index.html]

Fig 1: Allegro hand and Haptic Glove setup in Indian Institute of Technology Delhi,

The Allegro hand consists of three fingers and one thumb. Each of these fingers and thumbs has four actuators that give it the torque and angle of rotation it needs to hold an object. There are sixteen actuators in all. So when an operator is doing any activity with the haptic hand, which controls the allegro hand, we can get the force and angle. Force and angle, along with other parameters that depend on the object the Allegro hand interacts with, are used to train the machine learning model.



[ref:https://www.mdpi.com/2218-6581/8/4/86/htm, https://www.wonikrobotics.com/research-robot-hand]

Fig 2: Image of an Allegro Hand

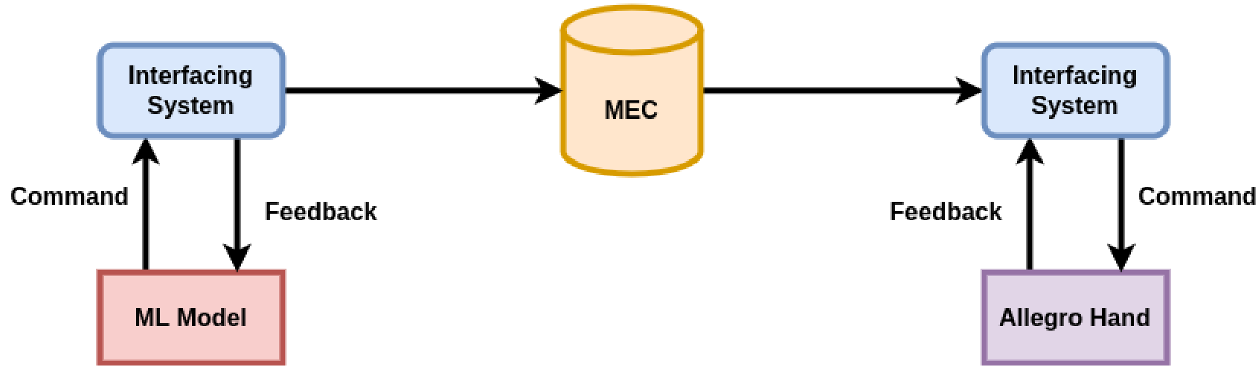


Fig 3: Use Of ML in controlling Allegro Hand

There are two use cases identified by the Indian Institute of Technology, Delhi based on the Low Latency Closed Loop between the haptic hands/ algorithms and real robotic hand (Allegro Hand) using MEC.

a. Slip Detection (and Force Estimation)

Given an object being held by an Allegro Hand. The object may tend to slip from the grip. Our aim is to predict if the object is slipping and Crumpling using ML models. So time-series data of the Allegro hand is provided, which have the force and angle made by the sixteen actuators of the allegro hand with the object along with the mass and the shape of the object. The values of slip and crumple can either be zero, which means the object is not slipping or crumpling. If the value is one, this means the object is slipping or crumpling. This data is provided for every instance of the given data. If an ML model can predict if the object is slipping and crumpling for every instance, then we can change the force and angle accordingly and make the hand balance the object. For now, the use case is to detect the slip and crumple. Later this approach can be extended to changing the force and angle to control the hand, so the object is not slipping or crumpling using ML. The use case of detecting slip and crumple is very useful for the task such as if the wire is being connected remotely using the Allegro hand, it might slip, catching an object if force is high object would be deformed etc.

b. Object Detection

The object is currently held by an Allegro hand. Our aim is to predict the shape of the object from 13 different shapes using an ML model. So the Allegro hand is provided, which has the force and angle created by the sixteen actuators of the allegro hand with the object along with the mass of the object. The shape of the object could be a sphere, cube, or cuboid with different dimensions. This is very useful as the allegro hand can easily grasp the object.

5 Design

Problem statement I: (Slip Detection and Force Estimation)

The initial datasets provided by the Indian Institute of Technology, Delhi, consist of timestamp data that predicts slip, crumple, or both. We were given the angle and force of 16 critical points on a hand, as well as the mass and shape of the object, and the objective was to train a model to predict whether it would slip or crumple.

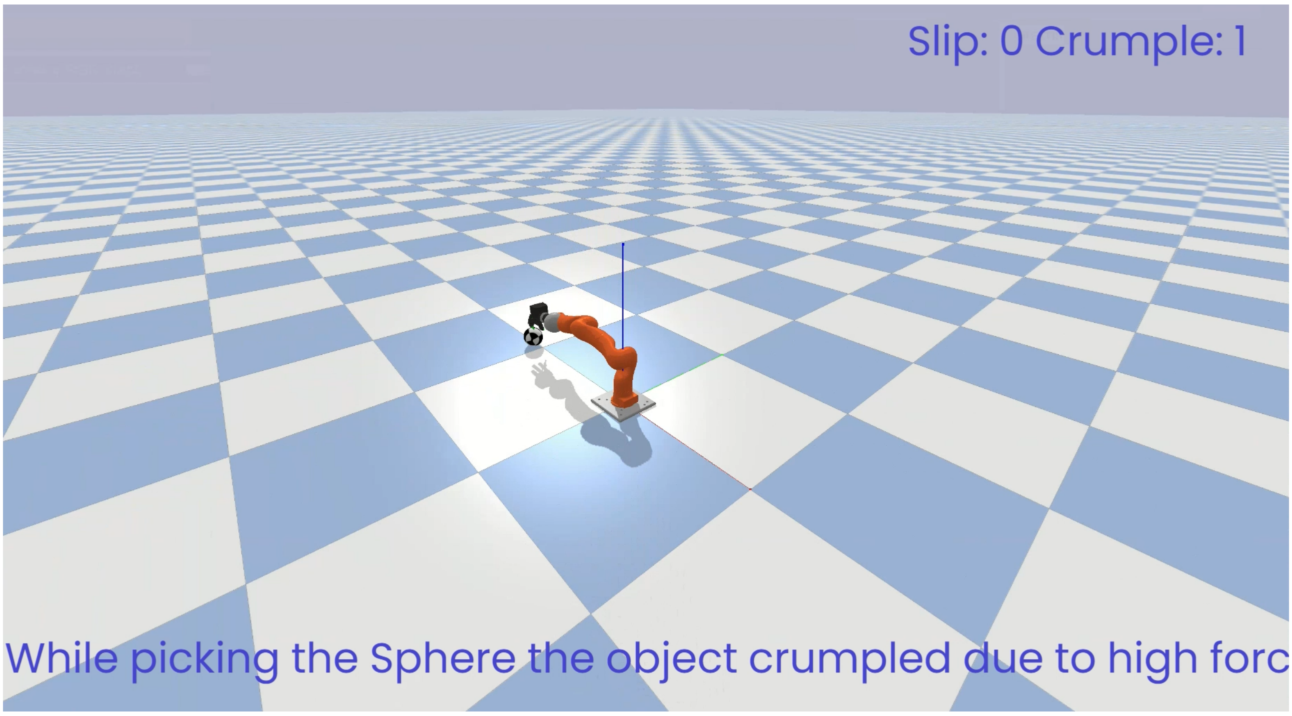


Fig 4: Sphere is crumbled from the grasp.

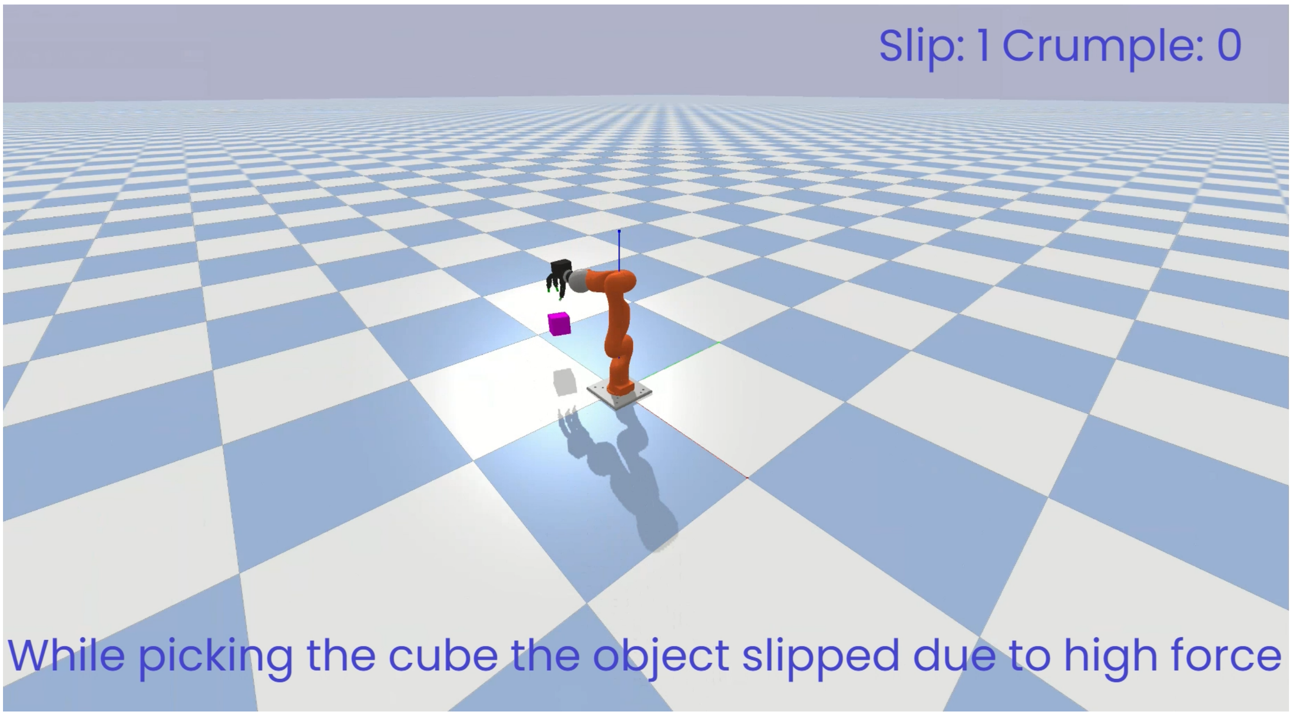


Fig 5: Cube slipped from the grasp.

Feature Engineering

The following are the steps we followed for feature engineering:

- The first thing we did was merge all the CSV files into a single file.
- Then we used a label encoder to encode the column named size as it contains alphabets, so we changed it to labels 0,1 and 2 for three object sizes, i.e., 5x10x5', 'R3.5' and '5x5x5.'
- Then we removed the timestamp column as we tried our approach both with and without the timestamp, and the latter method was more accurate.

  Note: The dataset provided uses different timestamps for each event which makes the impact of timestamps less related to the output (slip or crumple).

- Then we divide the data into X and Y, where X is the data given to the model and Y is the actual output label for slip and crumple.
- Then, we scale the data with the min-max scaler.
- At last, we split the data into a train and test set by shuffling the data and giving the data to the model.

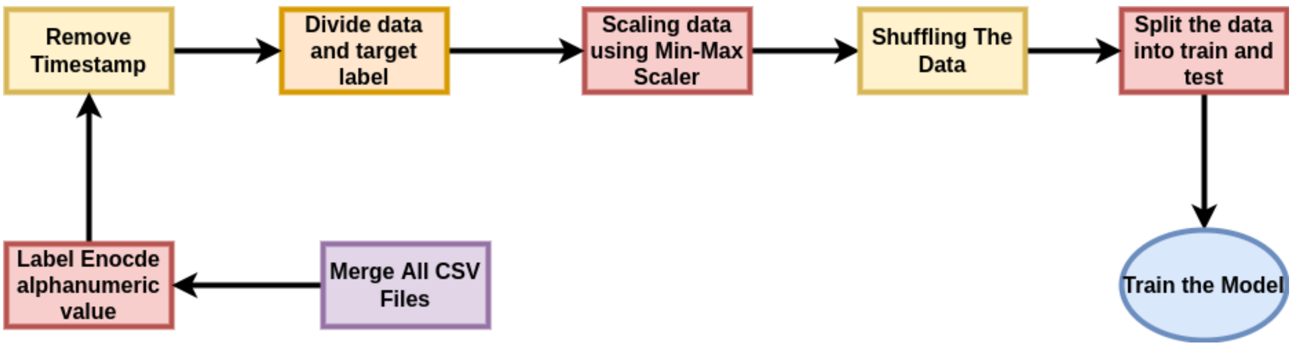


Fig6: Cleaning the Data for Problem Statement 1

Methodology

a) Experimental Methodologies Tried

We have tried out simple DNN and LSTM-based models to classify slip and crumple. The simple DNN failed as the slip and crumple for a particular object occurred continuously, which was not captured by the simple DNN model. Then we moved on to the LSTM-based model as the data was continuous, but LSTM failed as the data was changing the object continuously, so the relationships between different types of the object were hard to form. Then we chose a model which was able to convert the problem by finding more dominant features, which gave us a higher accuracy.

To classify slip and crumple, we used a Multitask Classifier. With the help of a feature transformer, an attentive transformer, and feature masking, this model encodes the features. Then, the input features are encoded in a fully connected layer to find the best and most dominant set of features. For reasoning, we use multiple decision blocks, each of which focuses on processing a different subset of the input features.

Our model consists of the following modules:

- Feature transformer is composed of a fully-connected (FC) layer, BN (Batch Normalization), and GLU (Gated Linear Unit) nonlinearity which converts features into more interpretable attributes.

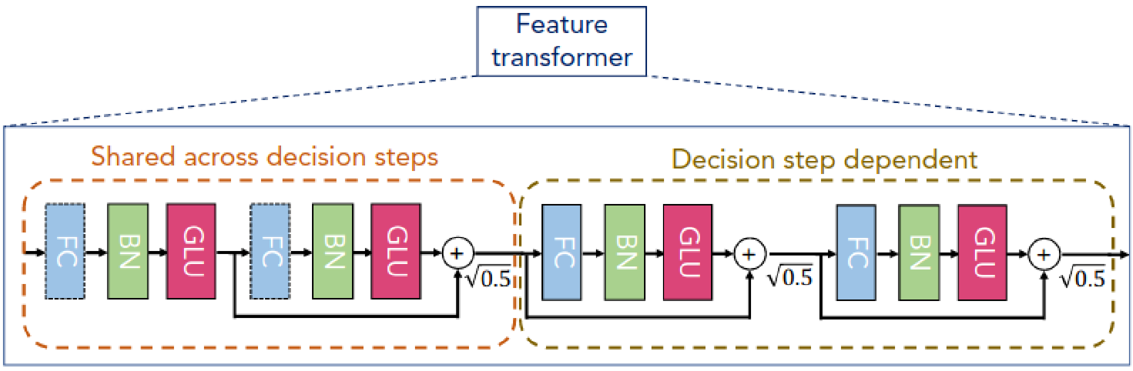


Fig7: Feature Transformer Block

- Attentive transformer blocks consist of a single layer mapping modulated with prior scale information, aggregating how much each feature has been used before the current decision step. Here, the coefficients are normalized using sparsemax, leading to a sparse selection of prominent features.

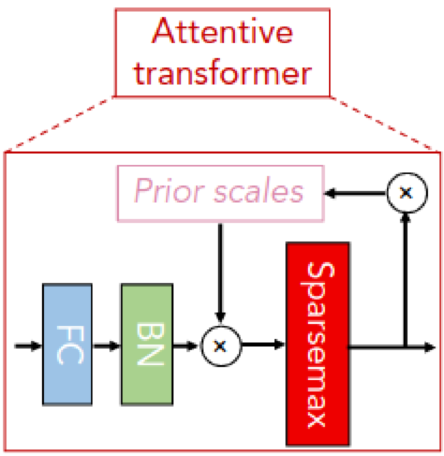


Fig8: Attentive Transformer

- Feature selection mask gives interpretable information about the operation of the model, and the masks can be used to get the required global feature attribution.

Input features are passed through the feature transformer and attentive transformer before finding the feature selection mask. Then, on finding the mask, the encoded features are again passed on to the feature transformer. A split block divides the processed representation to be used by the attentive transformer of the subsequent step as well as for the overall output, which is passed onto the Relu function. This is performed three times, and all the outputs of these layers are added and passed on to a fully connected layer.

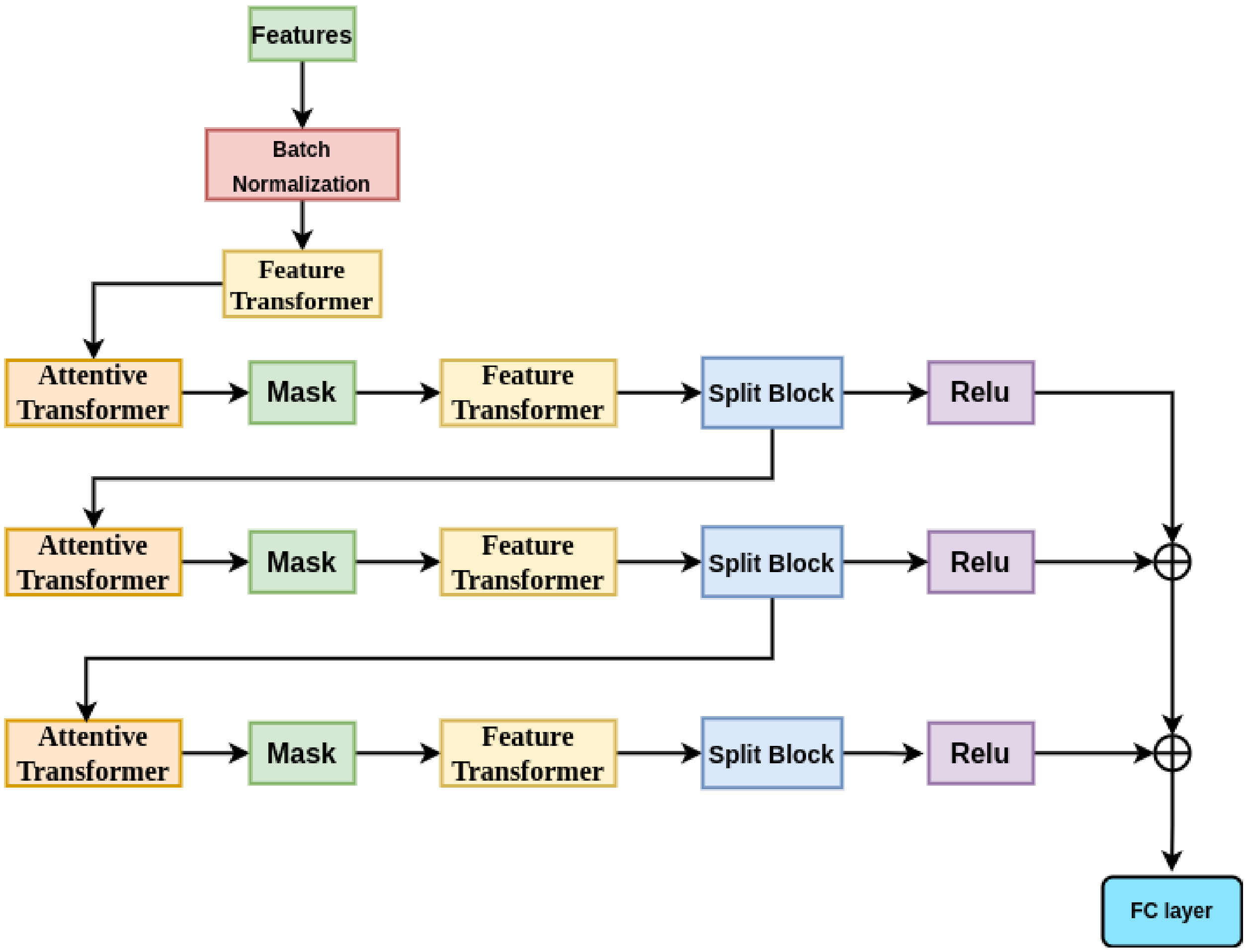


Fig 9: Architecture of proposed model for Problem Statement 1

Note: We have also used simple deep neural networks, LSTM, to train, but the accuracy of the proposed model was highest due to the reason that it encodes the features into a more meaningful entity, which correlates to the output labels more.

b) Hyperparameter Space (Learning rate, Model type, Optimization Scheme)

Framework used - Pytorch

Learning rate- 0.02

Optimizer - Adam

Loss Function - CrossEntropy

Momentum - 0.02

Model Type - Deep Neural Network

Epochs - 100

Validation Split - 0.16 percent of whole data

Batch Size - 2048

Note: We used 100 epochs as we had employed early stopping, and after 100 epochs the accuracy was not improving, rather it was going down.

c) Training Results

Models	**Proposed Model**	**DNN(5 layers)**	**DNN(10 layers)**	**LSTM**
Validation accuracy	**93.2**	86.2	87	89.4
Training Accuracy	**93.7**	87	87.5	90.1
Loss	**0.13521**	0.26	0.23	0.18

Table 1: Models tried for Problem Statement-1

Note: We have chosen the model with the highest accuracy and lowest loss.

Problem statement II: (Object Detection)

The initial datasets provided consist of data that predict the object's shape. The dataset includes the position and force of 16 key points on a hand, as well as the weight of the object. This data is provided to train a model to detect the shape of the object.

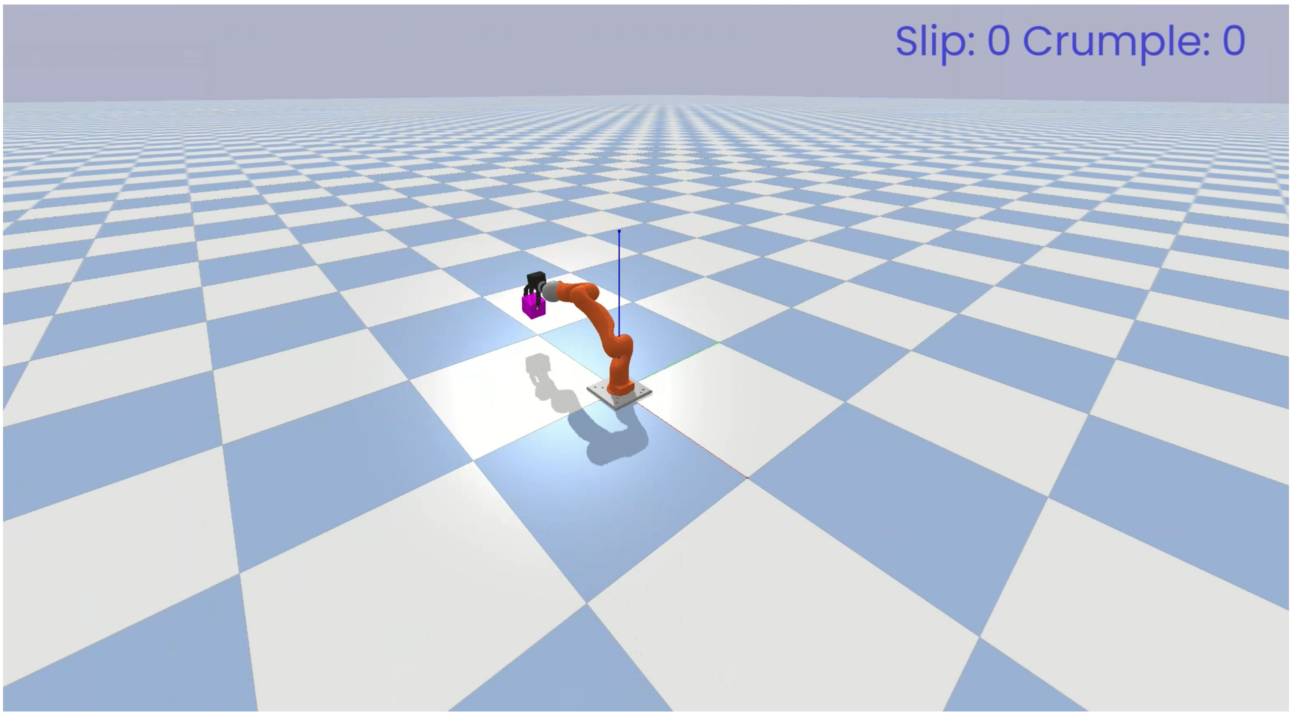


Fig 10: Hand Picked up the object to identify the shape as cube.

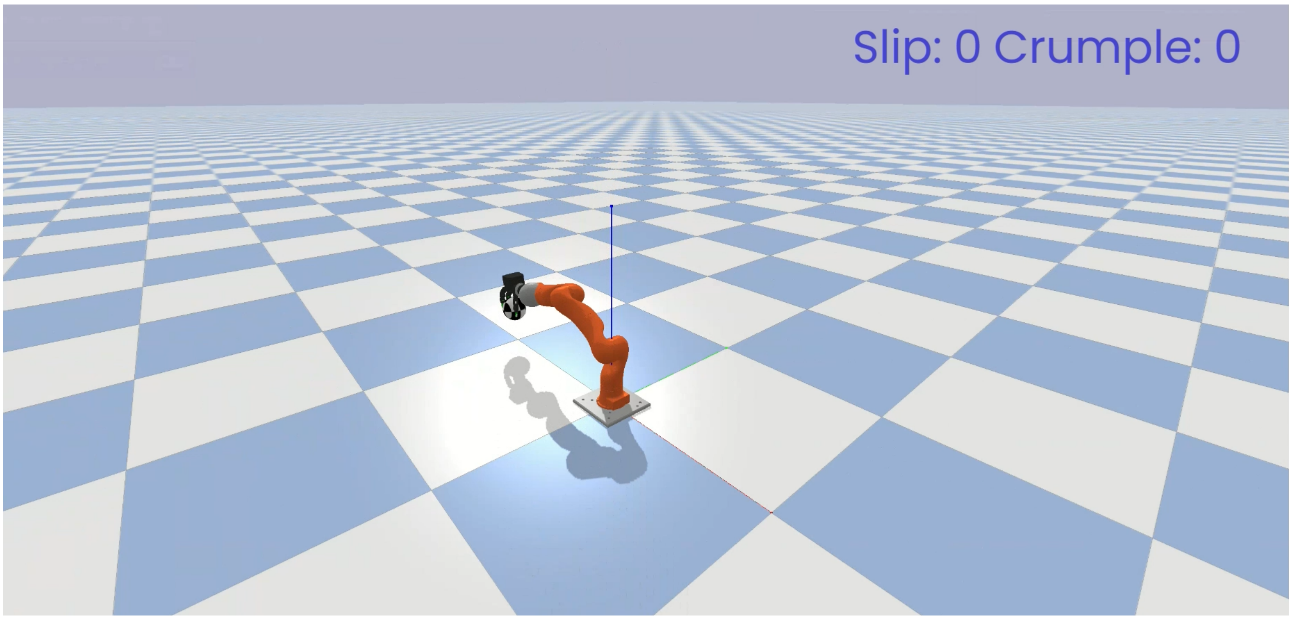


Fig 11: Hand Picked up the object to identify the shape as shpere.

Feature Engineering

Following are the steps we followed for Feature Engineering:

- First, we used a label encoder to encode the column Object_Held, which consists of the shape of the objects. It contains alphabets, so we have changed it to labels 0 to 12 for the 13 different objects' shapes.

- Then we split the data as X and Y, where X is the data that will be given to the model and Y is the actual output label of the object shape.
- Then, we scale the data with the min-max scaler
- Then we shuffle the data and split it before giving the data to the model.

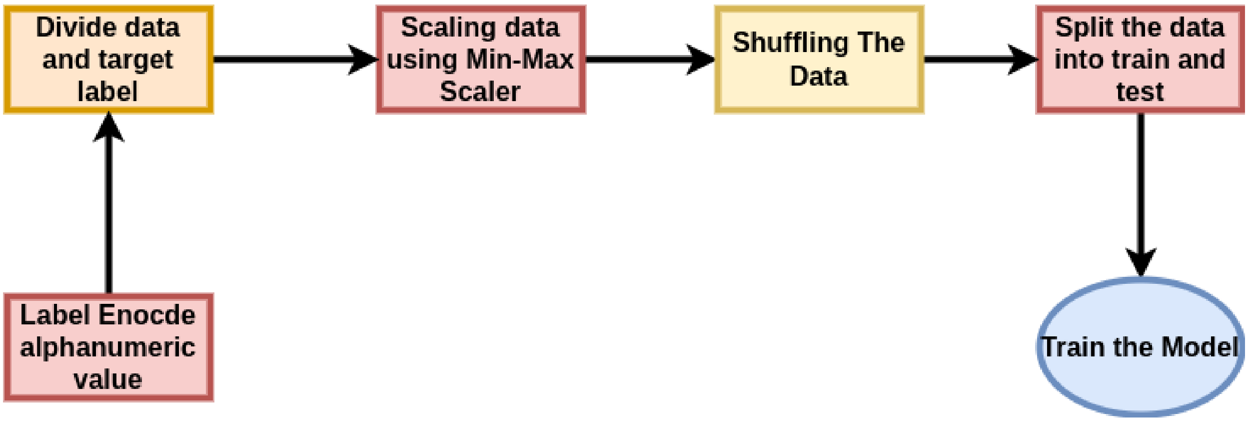


Fig 12: Cleaning the Data for Problem Statement 2

Methodology

a) Experimental Methodologies Tried

The model we used is a simple Deep Neural Network that consists of 5 fully connected (FC) layers. We have used 10-fold cross-validation to increase the accuracy and get the best parameters. We tried out other models but didn't get any drastic improvement, and this small model was giving similar results, so we went with the smaller model. To choose the final model, we tweaked the number of nodes in the layers and got the best result.

```
Net(
  (fc1): Linear(in_features=33, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=512, bias=True)
  (fc4): Linear(in_features=512, out_features=256, bias=True)
  (fc5): Linear(in_features=256, out_features=13, bias=True)
)
```

Fig 13: Architecture of proposed model for Problem Statement 2

b) Hyperparameter Space (Learning rate, Model type, Optimization Scheme)

Framework used - Pytorch

Learning rate- 0.002

Optimizer - Adam

Loss Function - CrossEntropy

Model Type - Deep Neural Network

Epochs - 200

Validation Split - 10 Fold validation

Batch Size - 2048

Note: We used 200 epochs as we had employed early stopping and after 100 epochs the accuracy was not improving, rather it was going down.

c) **Training Results**

Models	**Proposed Model**	**DNN(15 layers)**	**DNN(10 layers)**	**LSTM**
Validation accuracy	**99.7**	99.6	98.9	94.6
Training Accuracy	**99.8**	99.8	99.2	95.4
Loss	**0.014**	0.019	0.024	0.05

Table 2: Models tried for Problem Statement-2

Note: We chose a 5 layer deep DNN because it was a very simple architecture, but it was still performing very well compared to larger architectures.

6 Code and demo

```
        return x



```

In [5]:
```
def train_epoch(model,device,dataloader,loss_fn,optimizer):
    train_loss,train_correct=0.0,0
    model.train()
    for images, labels in dataloader:

        inputs,labels = images.to(device),labels.type(torch.LongTensor).to(device)
        optimizer.zero_grad()
        output = model(inputs)
        loss = loss_fn(output,labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * inputs.size(0)
        scores, predictions = torch.max(output.data, 1)
        train_correct += (predictions == labels).sum().item()

    return train_loss,train_correct

def valid_epoch(model,device,dataloader,loss_fn):
    valid_loss, val_correct = 0.0, 0
    model.eval()
    for images, labels in dataloader:

        inputs,labels = images.to(device),labels.type(torch.LongTensor).to(device)
        output = model(inputs)
        loss=loss_fn(output,labels)
        valid_loss+=loss.item()*inputs.size(0)
        scores, predictions = torch.max(output.data,1)
        val_correct+=(predictions == labels).sum().item()

    return valid_loss,val_correct
```

In [6]:
```

for fold, (train_idx,val_idx) in enumerate(splits.split(np.arange(len(loader)))):

    print('Fold {}'.format(fold + 1))

    train_sampler = SubsetRandomSampler(train_idx)
    test_sampler = SubsetRandomSampler(val_idx)
    train_loader = DataLoader(loader, batch_size=batch_size, sampler=train_sampler)
    test_loader = DataLoader(loader, batch_size=batch_size, sampler=test_sampler)

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    model = Net()
    model.to(device)
    optimizer = optim.Adam(model.parameters(), lr=0.002)

    history = {'train_loss': [], 'test_loss': [],'train_acc':[],'test_acc':[]}

    for epoch in range(num_epochs):
        train_loss, train_correct=train_epoch(model,device,train_loader,criterion,optimizer)
        test_loss, test_correct=valid_epoch(model,device,test_loader,criterion)

        train_loss = train_loss / len(train_loader.sampler)
        train_acc = train_correct / len(train_loader.sampler) * 100
        test_loss = test_loss / len(test_loader.sampler)
        test_acc = test_correct / len(test_loader.sampler) * 100

        print("Epoch:{}/{} AVG Training Loss:{:.3f} AVG Test Loss:{:.3f} AVG Training Acc {:.2f} % AVG Test Acc
                                                                                                              nu
                                                                                                              tr
                                                                                                              te
                                                                                                              tr
                                                                                                              te
        history['train_loss'].append(train_loss)
        history['test_loss'].append(test_loss)
        history['train_acc'].append(train_acc)
        history['test_acc'].append(test_acc)

    foldperf['fold{}'.format(fold+1)] = history
```

```
Epoch:6/250 AVG Training Loss:0.115 AVG Test Loss:0.066 AVG Training Acc 95.89 % AVG Test Acc 97.46 %
Epoch:7/250 AVG Training Loss:0.062 AVG Test Loss:0.092 AVG Training Acc 97.74 % AVG Test Acc 96.10 %
Epoch:8/250 AVG Training Loss:0.072 AVG Test Loss:0.123 AVG Training Acc 97.30 % AVG Test Acc 95.57 %
Epoch:9/250 AVG Training Loss:0.380 AVG Test Loss:0.096 AVG Training Acc 89.64 % AVG Test Acc 96.43 %
Epoch:10/250 AVG Training Loss:0.070 AVG Test Loss:0.061 AVG Training Acc 97.49 % AVG Test Acc 97.75 %
Epoch:11/250 AVG Training Loss:0.057 AVG Test Loss:0.051 AVG Training Acc 97.83 % AVG Test Acc 97.98 %
Epoch:12/250 AVG Training Loss:0.052 AVG Test Loss:0.053 AVG Training Acc 97.97 % AVG Test Acc 97.80 %
Epoch:13/250 AVG Training Loss:0.124 AVG Test Loss:0.051 AVG Training Acc 95.70 % AVG Test Acc 98.15 %
Epoch:14/250 AVG Training Loss:0.044 AVG Test Loss:0.048 AVG Training Acc 98.23 % AVG Test Acc 98.34 %
Epoch:15/250 AVG Training Loss:0.041 AVG Test Loss:0.039 AVG Training Acc 98.30 % AVG Test Acc 98.49 %
```

Fig 14: Notebook for problem statement-1

In [8]:
```
file=os.listdir('FORCE_ESTIMATION_TRAIN_SET')
data=pd.read_csv('FORCE_ESTIMATION_TRAIN_SET/1664273353.5152984.csv')
for i in file:
    merge=pd.read_csv("FORCE_ESTIMATION_TRAIN_SET/"+i)
    data=data.append(merge,ignore_index=True)
```

```
/tmp/ipykernel_100372/2747408133.py:5: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  data=data.append(merge,ignore_index=True)
/tmp/ipykernel_100372/2747408133.py:5: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  data=data.append(merge,ignore_index=True)
/tmp/ipykernel_100372/2747408133.py:5: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  data=data.append(merge,ignore_index=True)
/tmp/ipykernel_100372/2747408133.py:5: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  data=data.append(merge,ignore_index=True)
/tmp/ipykernel_100372/2747408133.py:5: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  data=data.append(merge,ignore_index=True)
/tmp/ipykernel_100372/2747408133.py:5: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  data=data.append(merge,ignore_index=True)
/tmp/ipykernel_100372/2747408133.py:5: FutureWarning: The frame.append method is deprecated and will be removed
```

In [9]:
```
data
```

Out[9]:

	Timestamp	J0_position	J0_Force	J1_position	J1_Force	J2_position	J2_Force	J3_position	J3_Force	J4_position	...	J13_position	J13_For
0	1.664273e+09	0.040519	81.392987	0.798471	81.385970	0.586294	81.357743	0.034970	46.121681	0.020831	...	-0.277912	151.4922
1	1.664273e+09	0.041405	83.821461	0.800734	83.807202	0.579219	83.739365	0.044107	50.646438	0.021107	...	-0.264644	159.8904
2	1.664273e+09	0.041906	76.073889	0.803809	76.049124	0.568351	75.923921	0.060751	52.002144	0.020887	...	-0.256384	155.1788
3	1.664273e+09	0.041901	73.645277	0.805177	73.627524	0.570574	73.539115	0.057264	50.186518	0.020778	...	-0.249897	152.9471
4	1.664273e+09	0.041857	76.008189	0.806973	76.008990	0.571762	76.018871	0.053316	48.206326	0.020751	...	-0.236733	158.2622
...	...	...	...	...	...	...	...	...	...	...	...	...	
467745	1.664304e+09	6.463620	34.886219	7.742457	34.911721	7.265793	18.622628	6.346061	18.712875	8.157747	...	6.438690	66.5115
467746	1.664304e+09	6.463641	35.356901	7.742312	35.381891	7.266195	19.428752	6.345851	19.597824	8.157777	...	6.438726	66.2826
467747	1.664304e+09	6.463647	34.659822	7.742252	34.685034	7.266330	18.965942	6.345789	19.079189	8.157778	...	6.438764	66.4750
467748	1.664304e+09	6.463598	34.936214	7.742506	34.961984	7.265580	18.673088	6.346264	18.752918	8.157736	...	6.438830	66.4467
467749	1.664304e+09	6.463599	34.912609	7.742538	34.938348	7.265547	18.671525	6.346255	18.751086	8.157738	...	6.438838	66.4484

467750 rows × 37 columns

In [12]:
```
label_encoder = preprocessing.LabelEncoder()
data["Size"]= label_encoder.fit_transform(data["Size"])
min_max_scaler = MinMaxScaler()
x = data.iloc[:,1:35].values
#x = min_max_scaler.fit_transform(x)
y = data.iloc[:,35:37].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.16, random_state=42)
model= TabNetMultiTaskClassifier()
```

```
/home/server/anaconda3/lib/python3.9/site-packages/pytorch_tabnet/abstract_model.py:75: UserWarning: Device used :
cuda
  warnings.warn(f"Device used : {self.device}")
```

In [13]:
```
x
```

Out[13]:
```
array([[4.05190891e-02, 8.13929866e+01, 7.98471418e-01, ...,
        1.45388558e+02, 0.00000000e+00, 2.00000000e+00],
       [4.14052861e-02, 8.38214614e+01, 8.00734230e-01, ...,
        1.26929915e+02, 0.00000000e+00, 2.00000000e+00],
       [4.19064364e-02, 7.60738887e+01, 8.03808660e-01, ...,
        1.27926552e+02, 0.00000000e+00, 2.00000000e+00],
       ...,
       [6.46364688e+00, 3.46598215e+01, 7.74225245e+00, ...,
        6.62412028e+01, 2.00000000e+00, 1.00000000e+00],
       [6.46359764e+00, 3.49362144e+01, 7.74250600e+00, ...,
        6.62188541e+01, 2.00000000e+00, 1.00000000e+00],
       [6.46359855e+00, 3.49126092e+01, 7.74253801e+00, ...,
        6.62209183e+01, 2.00000000e+00, 1.00000000e+00]])
```

In [14]:
```
model= TabNetMultiTaskClassifier()
```

In [15]:
```
model
```

Out[15]:
```
TabNetMultiTaskClassifier(n_d=8, n_a=8, n_steps=3, gamma=1.3, cat_idxs=[], cat_dims=[], cat_emb_dim=1, n_independe
nt=2, n_shared=2, epsilon=1e-15, momentum=0.02, lambda_sparse=0.001, seed=0, clip_value=1, verbose=1, optimizer_fn
=<class 'torch.optim.adam.Adam'>, optimizer_params={'lr': 0.02}, scheduler_fn=None, scheduler_params={}, mask_type
='sparsemax', input_dim=None, output_dim=None, device_name='auto', n_shared_decoder=1, n_indep_decoder=1)
```

In [16]:
```

model.fit(
    x_train,y_train,
    eval_set=[(x_train, y_train), (x_test, y_test)],
    eval_name=['train', 'valid'],
    eval_metric=['accuracy'],
    max_epochs=100,batch_size=2048
)


saved_filepath = model.save_model(args.model_path+"/model_checkpoint_task1")
```

```
epoch 0  | loss: 0.19779 | train_accuracy: 0.9148  | valid_accuracy: 0.91537 |  0:00:15s
epoch 1  | loss: 0.16437 | train_accuracy: 0.9183  | valid_accuracy: 0.91836 |  0:00:30s
epoch 2  | loss: 0.16061 | train_accuracy: 0.91897 | valid_accuracy: 0.9188  |  0:00:45s
epoch 3  | loss: 0.16046 | train_accuracy: 0.91948 | valid_accuracy: 0.9198  |  0:00:59s
epoch 4  | loss: 0.15697 | train_accuracy: 0.91989 | valid_accuracy: 0.91966 |  0:01:14s
epoch 5  | loss: 0.1553  | train_accuracy: 0.92132 | valid_accuracy: 0.92173 |  0:01:29s
epoch 6  | loss: 0.15506 | train_accuracy: 0.92061 | valid_accuracy: 0.9208  |  0:01:43s
epoch 7  | loss: 0.15423 | train_accuracy: 0.9217  | valid_accuracy: 0.92209 |  0:01:58s
```

Fig 15: Notebook for problem statement-2

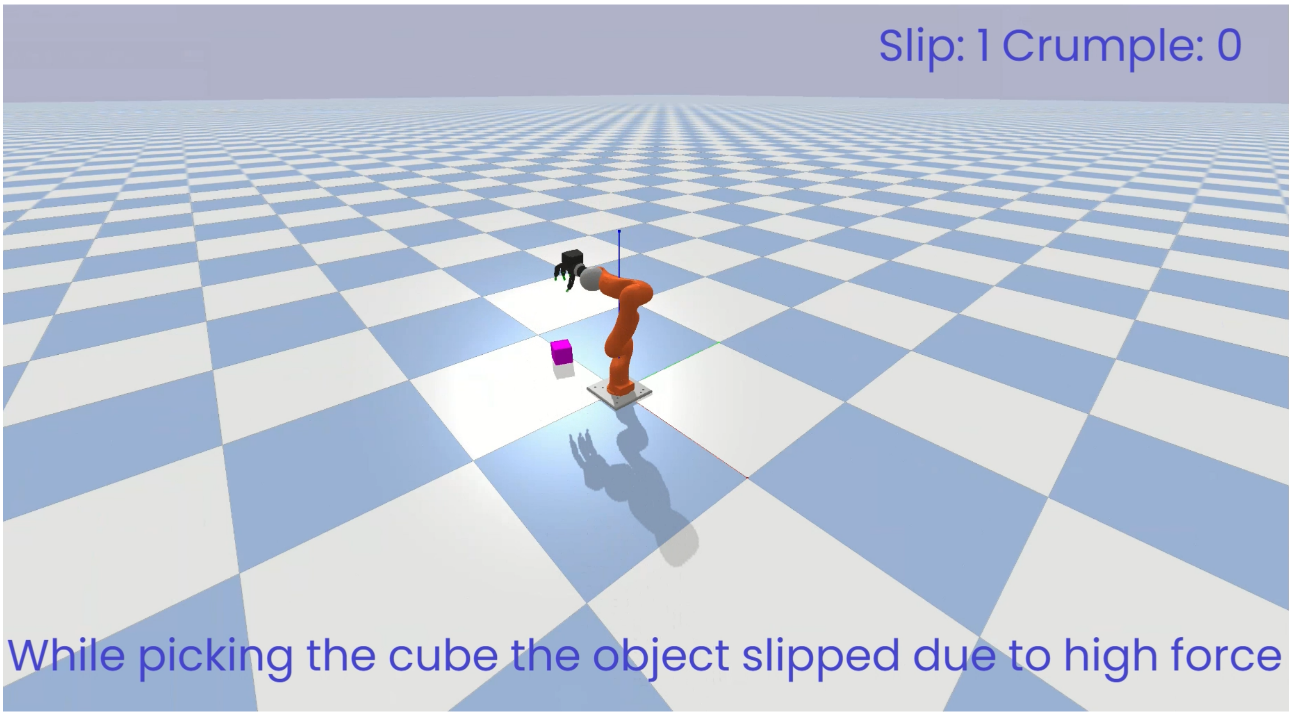


Fig 16: Cube slipped from the grasp.

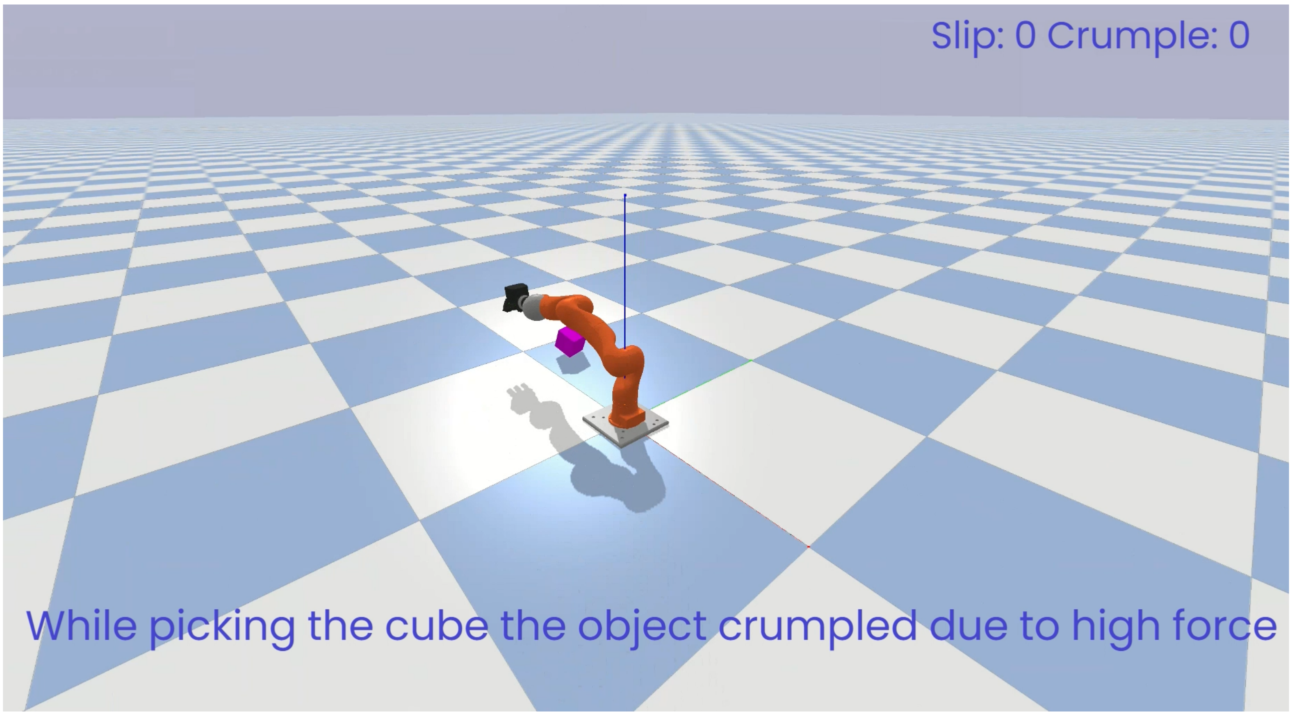


Fig 17: Cube crumpling from the grasp.

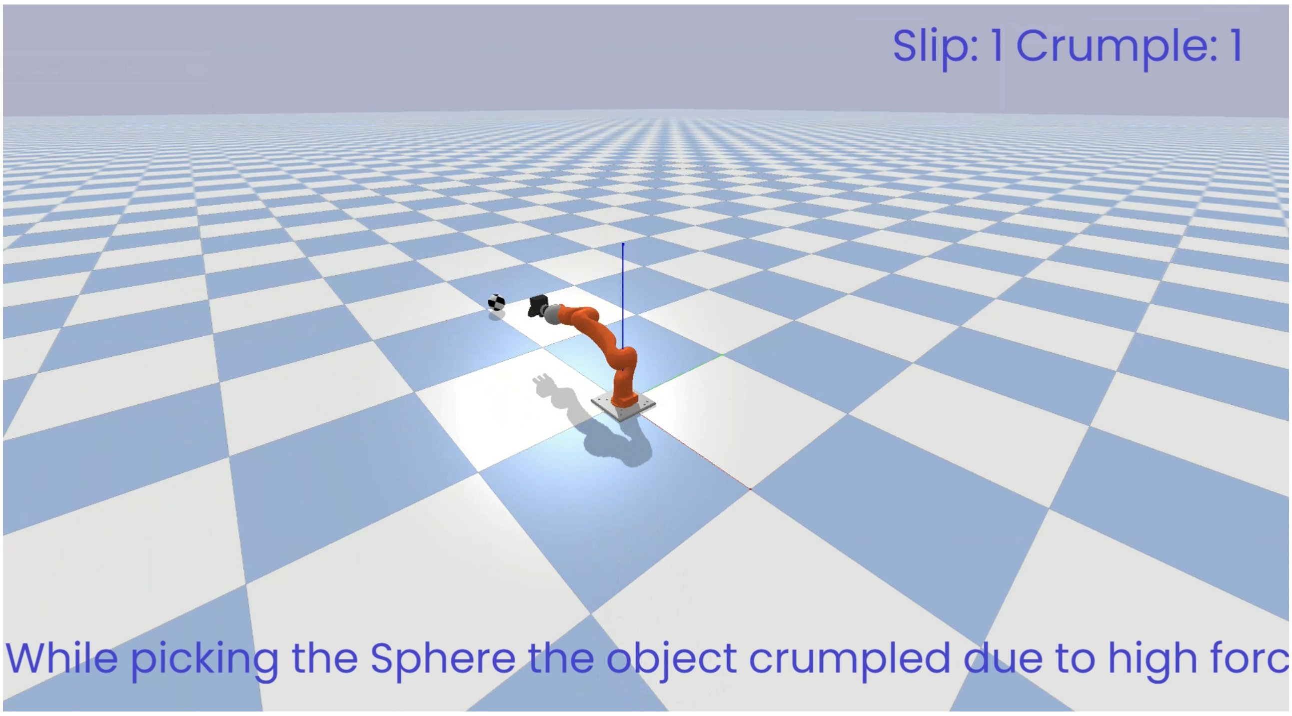


Fig 18: Sphere crumpled from the grasp.

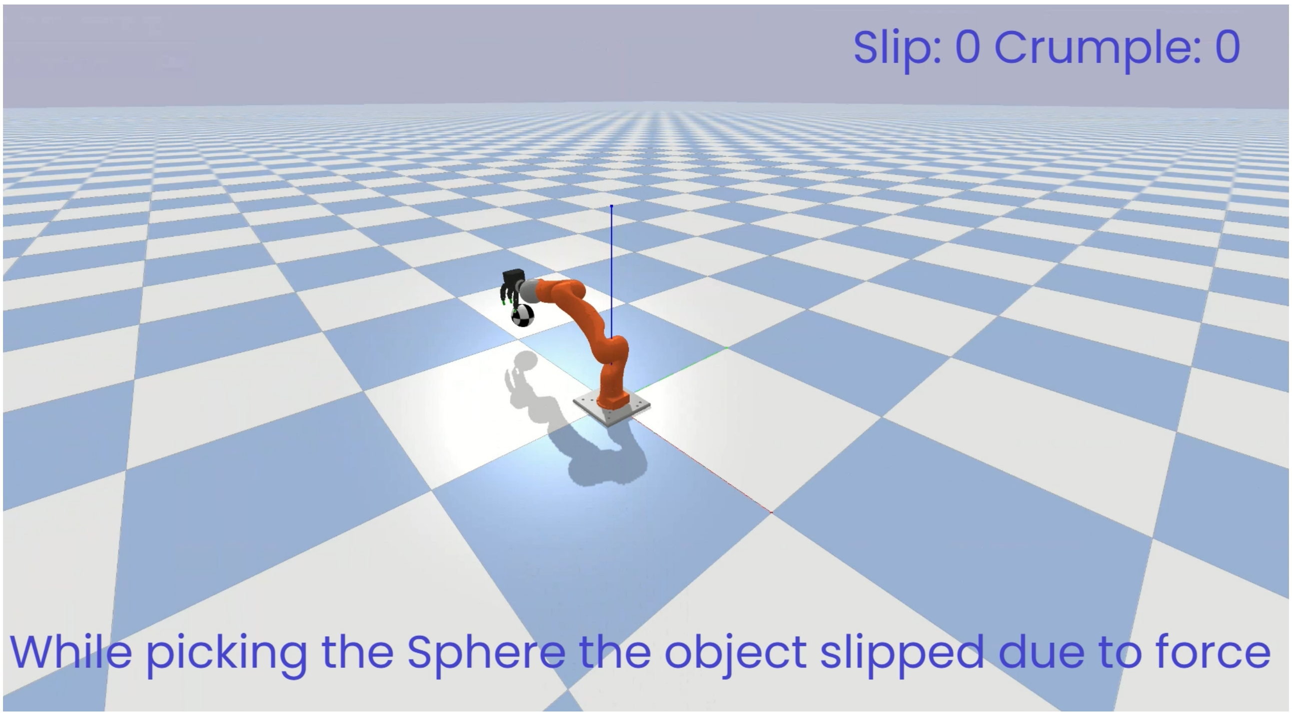


Fig 19: Sphere Slipped from the grasp.

```
app.create_actors_relationship_with_usecase("PS1 model", "outputs", "Slip ", "usecase_001")
app.create_actors_relationship_with_usecase("PS1 model", "outputs", "Crumple ", "usecase_001")
app.create_actors_relationship_with_usecase("Force OF Joints", "Inputs", "PS1 model", "usecase_001")
app.create_actors_relationship_with_usecase("Mass of object", "Inputs", "PS1 model", "usecase_001")
app.create_actors_relationship_with_usecase("Shape of Object", "Inputs", "PS1 model", "usecase_001")
app.create_actors_relationship_with_usecase("Time stamp", "Inputs", "PS1_model", "usecase_001")



app.create_actors_relationship_with_usecase("MEC", "outputs", "Time stamp", "usecase_001")
app.create_actors_relationship_with_usecase("MEC", "outputs", "Shape of Object", "usecase_001")
app.create_actors_relationship_with_usecase("MEC", "outputs", "Mass of object", "usecase_001")
app.create_actors_relationship_with_usecase("MEC", "outputs", "Force OF Joints", "usecase_001")



app.create_actors_relationship_with_usecase("Haptic Arm", "Inputs", "MEC", "usecase_001")
app.create_actors_relationship_with_usecase("Object", "Inputs", "MEC", "usecase_001")



app.create_actors_relationship_with_usecase("PS2 model", "outputs", "Shape of Object ", "usecase_002")
app.create_actors_relationship_with_usecase("Force OF Joints", "Inputs", "PS2 model", "usecase_002")
app.create_actors_relationship_with_usecase("Mass of object", "Inputs", "PS2 model", "usecase_002")



app.create_actors_relationship_with_usecase("MEC", "outputs", "Mass of object", "usecase_002")
app.create_actors_relationship_with_usecase("MEC", "outputs", "Force OF Joints", "usecase_002")



app.create_actors_relationship_with_usecase("Haptic Arm", "Inputs", "MEC", "usecase_002")
app.create_actors_relationship_with_usecase("Object", "Inputs", "MEC", "usecase_002")
```

Note: Code for the graph generation.

Problem statement I: (Slip Detection and Force Estimation)

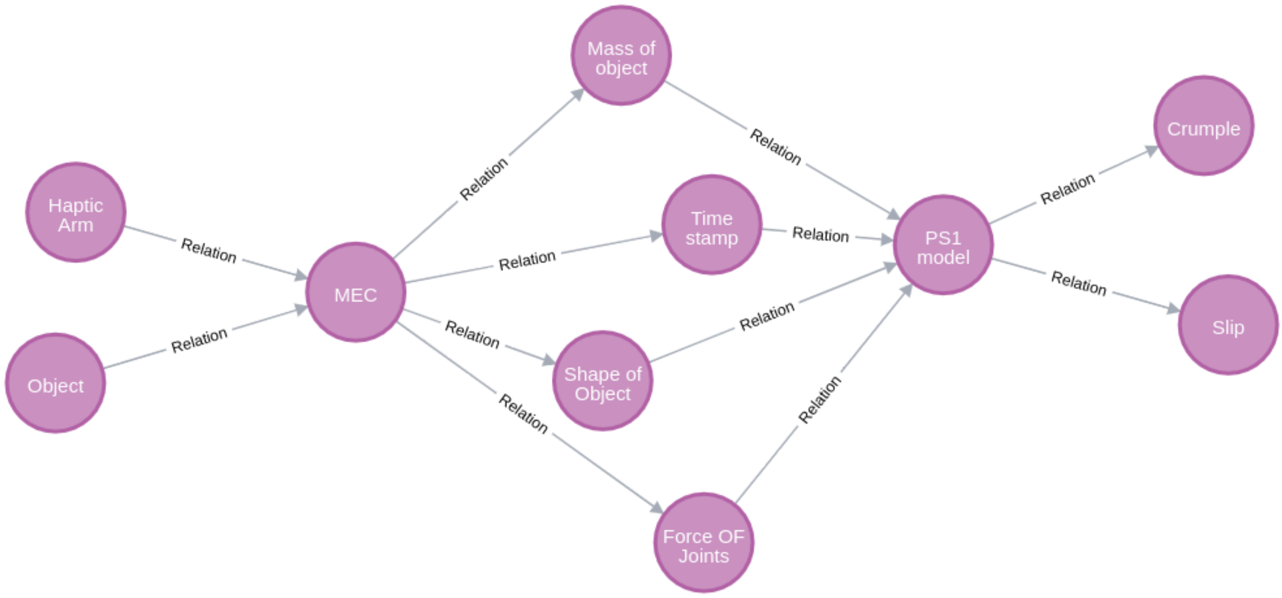


Fig 20: Graph for problem statement-1

Actor	Description	Relation
Haptic Arm	Gives force and angle to hold a object	Haptic Arm -> Inputs -> MEC
Object	Shape and mass of Object	Object -> Inputs -> MEC
MEC	Outputs the haptic arm and object information to the model	MEC-> outputs -> Time stamp
		MEC-> outputs -> Shape of Object
		MEC-> outputs -> Mass of object
		MEC-> outputs -> Force OF Joints
Mass of Object	Mass of the Object	Mass of object-> Inputs -> PS1 model
Time Stamp	Time of event	Time Stamp -> Inputs -> PS1 model
Shape of object	Shape of the object	Shape of object Inputs -> PS1 model
Force of Joints	Force and angle made by haptic arm on object	Force OF Joints Inputs -> PS1 model
PS1 Model	Model which predict slip or crumple	PS1 model -> outputs -> slip
		PS1 model -> outputs -> crumple

Table 3: Relationship table for Problem Statement-1

Problem statement II: (Object Detection)

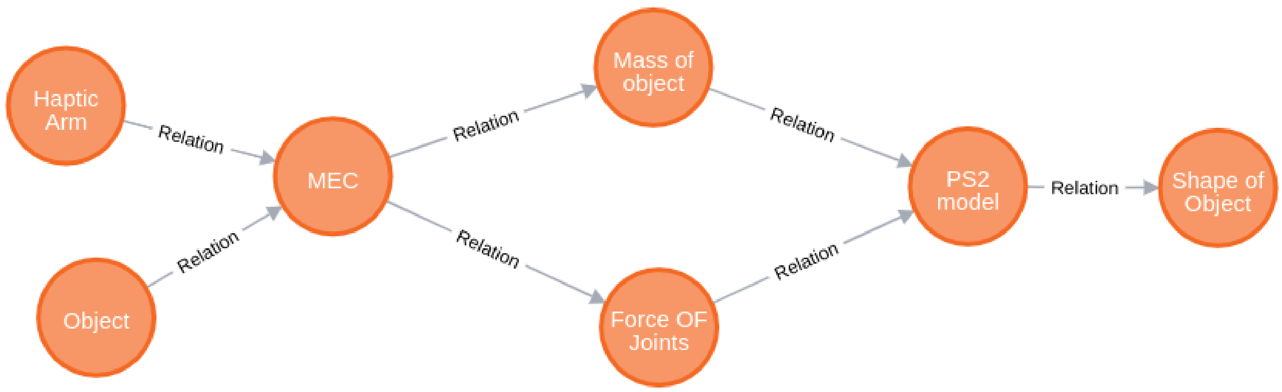


Fig 21: Graph for problem statement-2

Actor	Description	Relation
Haptic Arm	Gives force and angle to hold a object	Haptic Arm -> Inputs -> MEC
Object	Shape and mass of Object	Object -> Inputs -> MEC
MEC	Outputs the haptic arm and object information to the model	MEC-> outputs -> Mass of object
		MEC-> outputs -> Force OF Joints
Mass of Object	Mass of the Object	Mass of object-> Inputs -> PS2 model
Force of Joints	Force and angle made by haptic arm on object	Force OF Joints Inputs -> PS2 model
PS2 Model	Model which predict slip or crumple	PS1 model -> outputs -> shape of object

Table 4: Relationship table for Problem Statement-2

7 Summary

In this document, we discussed the use case from Indian Institute of Technology Delhi and analyzed the design. We provide extensions to the reference code provided in [Build-a-thon 2022] and build our own graph based on the reference code.

The report includes the following:

- Analysis of the use case with examples
- A design of the use case
- Code to produce the graph-based design based on neo4j as per the reference code provided in the Build-a-thon repo.
- Demo Video (with caption)
- Screenshots of Models and simulator
- ML Model architectures used
- Accuracy and hyperparameters of the Models