B.Tech Mini Project Report

Hemang Joshi (202001212)

Under the guidance of

Prof. Amit Mankodi

1. OBJECTIVE:

To predict the runtime of a code which is based on the evolution and propagation of Plasma in a 2D space with a defined area of interest using Machine Learning Techniques. The code which simulates the propagation of Plasma takes a very long time to run, and hence it becomes of great help to know priorly how much time would the code take to run on a given system with a given set of configurations.

The code was a C language code and had a defined set of parameters which can be varied to test the runtime of the code. Apart from that the code has to be tested on different CPUs with a diverse set of configurations which would help in predicting the runtime.

To achieve the objective, first the runtime data had to be cleaned and analyzed. Also, it was expected to identify the dominating factors which affect the runtime greatly. This can help in designing a custom CPU with a favorable condition for this specific application purpose.

2. SCOPE & ASSUMPTIONS:

SCOPE:

- Predict the runtime of the evolution and propagation of Plasma code.
- Use Machine Learning Techniques to predict the runtime.
- Identify dominant factors which affect the runtime performance of the code.
- For performing the above tasks, analyze and clean the input data in a suitable manner to feed the model.
- Generate the dataset by testing the code on different systems each with a different set of configurations.

ASSUMPTIONS:

- The dataset is compiled on idle systems with almost no processes running in the background except the necessary system processes.
- It is assumed that changing the operating system would not affect the runtime or does not require any kind of optimization.
- The code uses a variable named KELEC, which is proportional to the time of propagation of Plasma in space. This variable was fixed at a constant value of 10, to speed up the process of generating the dataset. This means that for all the data points, the duration of propagation of Plasma was fixed.

3. TOOLS & TECHNOLOGIES:

- C Language
- Shell commands
- Python and related libraries (Numpy, Pandas, Matplotlib, Seaborn)
- Machine Learning and related libraries (Scikit-learn)
- Basics of Deep Learning with Neural Networks (Keras)
- Google Colab

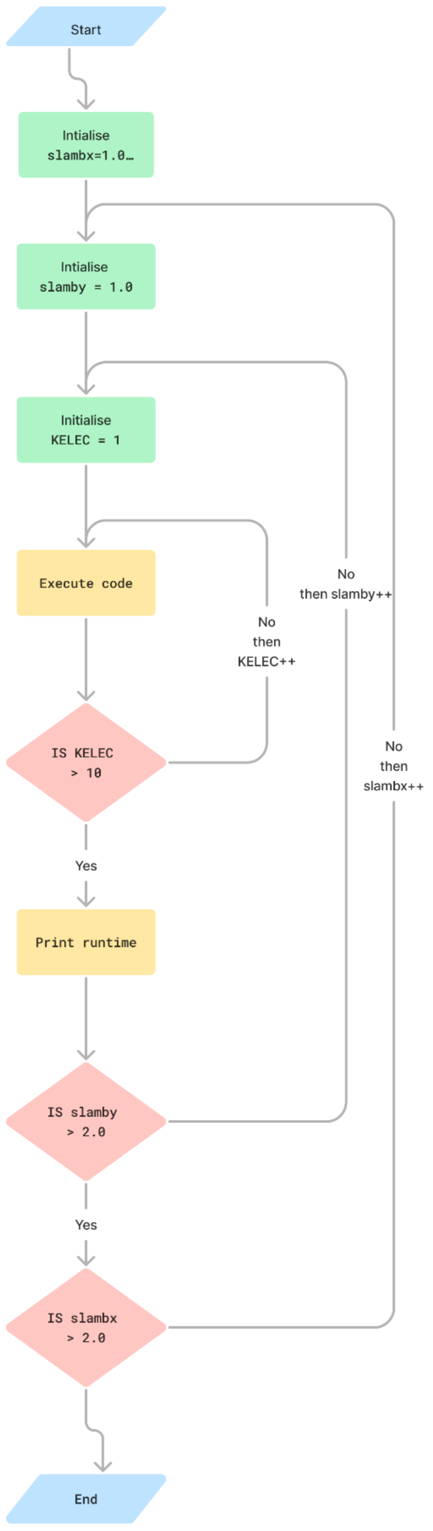
4. WORKFLOW:

Flowchart of the C code indicating the modifications done to extract the data.

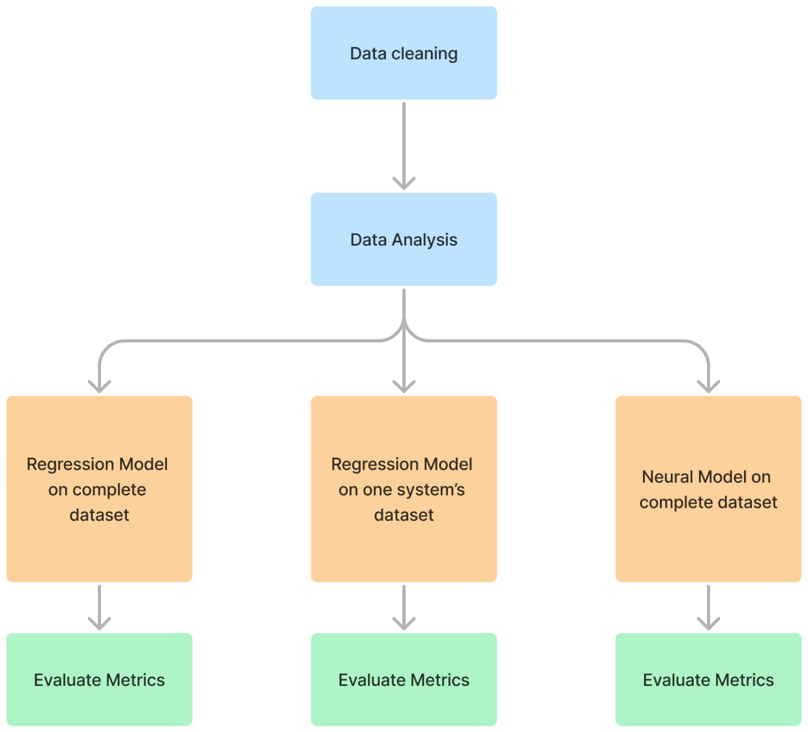
The slambx and slamby are parameters defining the area of interest for the propagation of the plasma.

The KELEC parameter is proportional to the duration of the propagation.

The output is the time taken by the code to run for a given value of slambx and slamby on the particular system, where slambx and slamby both vary from 1.0 to 1.9, generating 100 data points for each system.

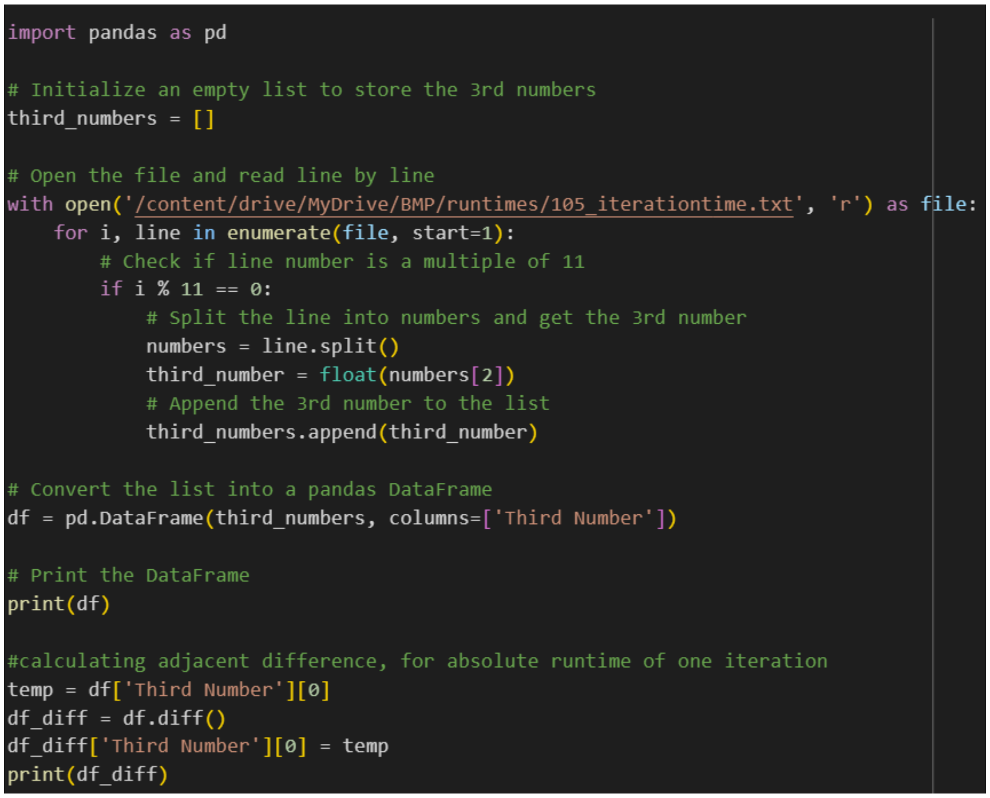


Data Analysis and Regression Flow


Data cleaning
Data Analysis
Regression Model on complete dataset
Regression Model on one system's dataset
Neural Model on complete dataset
Evaluate Metrics
Evaluate Metrics
Evaluate Metrics

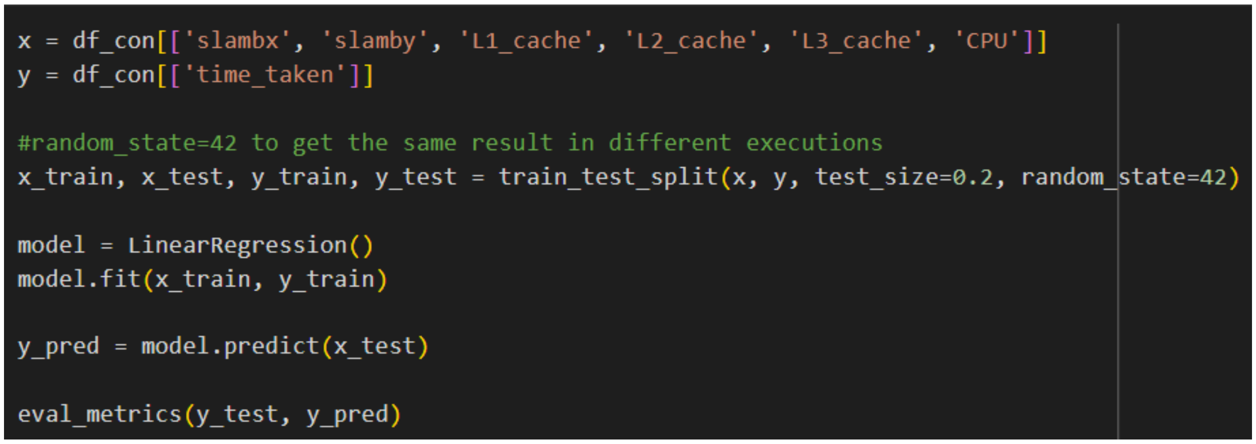
5. CODE SNIPPETS:

Cleaning and Formatting the input raw data into a csv format.



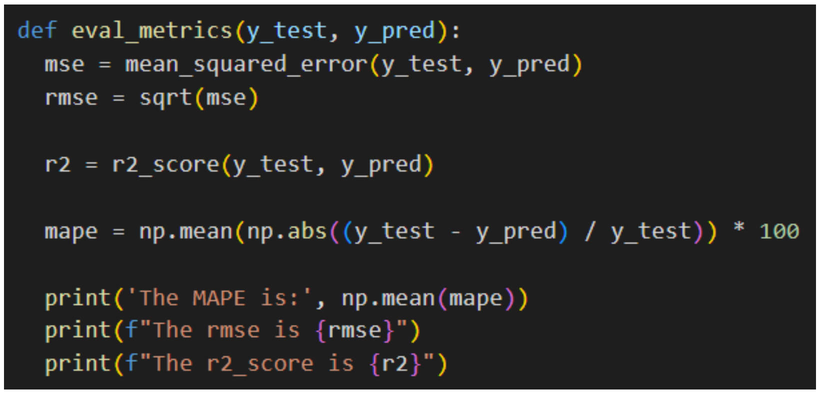
```python
import pandas as pd

# Initialize an empty list to store the 3rd numbers
third_numbers = []

# Open the file and read line by line
with open('/content/drive/MyDrive/BMP/runtimes/105_iterationtime.txt', 'r') as file:
    for i, line in enumerate(file, start=1):
        # Check if line number is a multiple of 11
        if i % 11 == 0:
            # Split the line into numbers and get the 3rd number
            numbers = line.split()
            third_number = float(numbers[2])
            # Append the 3rd number to the list
            third_numbers.append(third_number)

# Convert the list into a pandas DataFrame
df = pd.DataFrame(third_numbers, columns=['Third Number'])

# Print the DataFrame
print(df)

#calculating adjacent difference, for absolute runtime of one iteration
temp = df['Third Number'][0]
df_diff = df.diff()
df_diff['Third Number'][0] = temp
print(df_diff)
```

Linear Regression Model



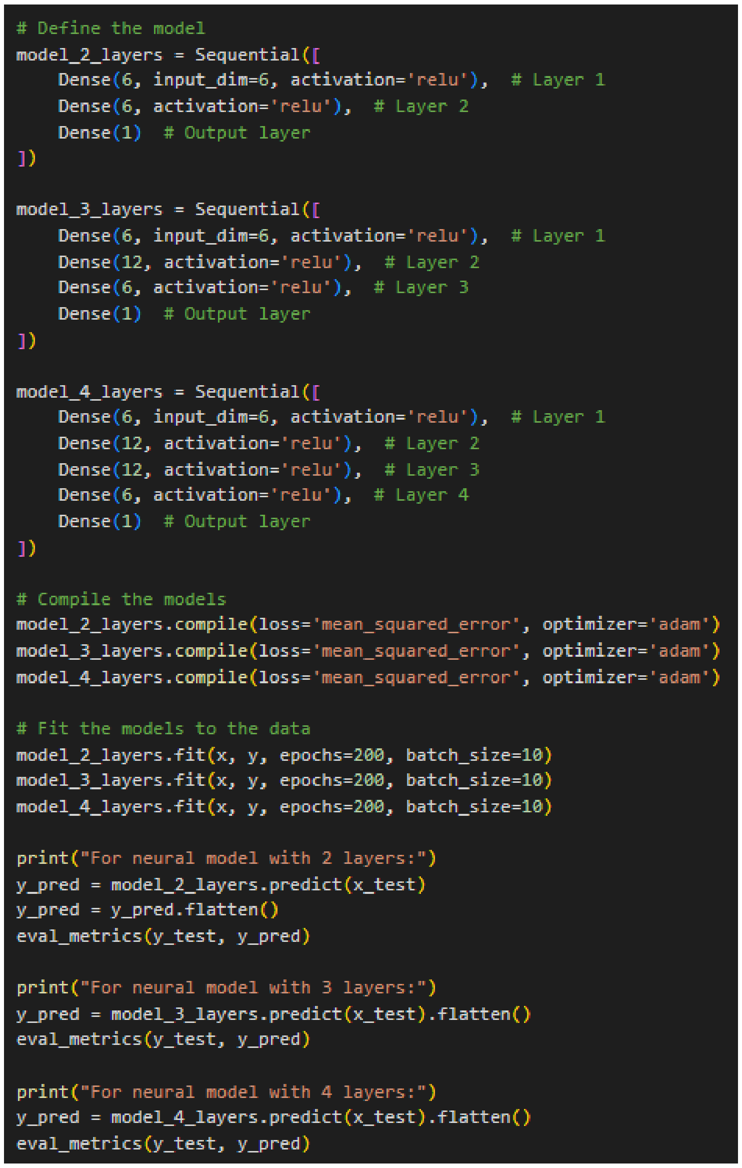
```python
x = df_con[['slambx', 'slamby', 'L1_cache', 'L2_cache', 'L3_cache', 'CPU']]
y = df_con[['time_taken']]

#random_state=42 to get the same result in different executions
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

eval_metrics(y_test, y_pred)
```

Evaluation of Metrics function



```python
def eval_metrics(y_test, y_pred):
  mse = mean_squared_error(y_test, y_pred)
  rmse = sqrt(mse)

  r2 = r2_score(y_test, y_pred)

  mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

  print('The MAPE is:', np.mean(mape))
  print(f"The rmse is {rmse}")
  print(f"The r2_score is {r2}")
```

Neural Model



```python
# Define the model
model_2_layers = Sequential([
    Dense(6, input_dim=6, activation='relu'),  # Layer 1
    Dense(6, activation='relu'),  # Layer 2
    Dense(1)  # Output layer
])

model_3_layers = Sequential([
    Dense(6, input_dim=6, activation='relu'),  # Layer 1
    Dense(12, activation='relu'),  # Layer 2
    Dense(6, activation='relu'),  # Layer 3
    Dense(1)  # Output layer
])

model_4_layers = Sequential([
    Dense(6, input_dim=6, activation='relu'),  # Layer 1
    Dense(12, activation='relu'),  # Layer 2
    Dense(12, activation='relu'),  # Layer 3
    Dense(6, activation='relu'),  # Layer 4
    Dense(1)  # Output layer
])

# Compile the models
model_2_layers.compile(loss='mean_squared_error', optimizer='adam')
model_3_layers.compile(loss='mean_squared_error', optimizer='adam')
model_4_layers.compile(loss='mean_squared_error', optimizer='adam')

# Fit the models to the data
model_2_layers.fit(x, y, epochs=200, batch_size=10)
model_3_layers.fit(x, y, epochs=200, batch_size=10)
model_4_layers.fit(x, y, epochs=200, batch_size=10)

print("For neural model with 2 layers:")
y_pred = model_2_layers.predict(x_test)
y_pred = y_pred.flatten()
eval_metrics(y_test, y_pred)

print("For neural model with 3 layers:")
y_pred = model_3_layers.predict(x_test).flatten()
eval_metrics(y_test, y_pred)

print("For neural model with 4 layers:")
y_pred = model_4_layers.predict(x_test).flatten()
eval_metrics(y_test, y_pred)
```

6. RESULTS:

For one system, the best score is mentioned out of all the datasets.

	R^2 score	RMSE	MAPE
Linear Regression on complete dataset	0.9138	69.7945	7.6136
Linear Regression on one system's dataset	0.9562	32.3571	5.5828

Neural Model:

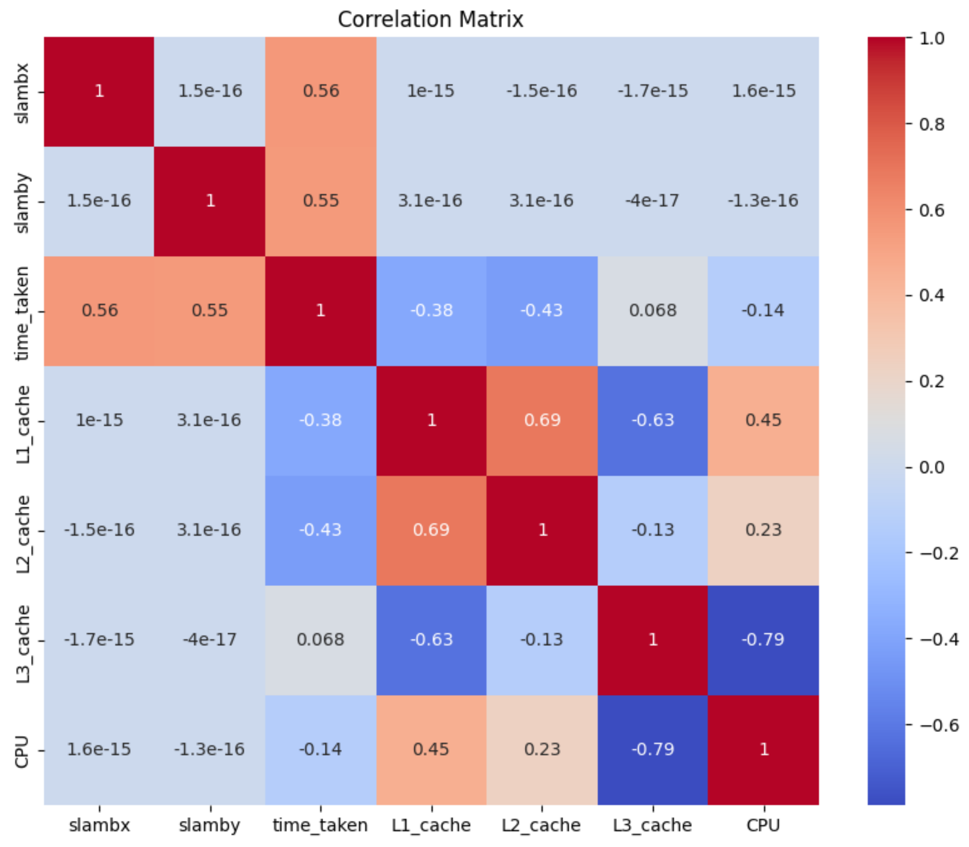
In the 2-layer neural model, the hidden layer had 2 layers with 6 neurons each.

In the 3-layer neural model, the hidden layer had 3 layers with 6, 12, 6 neurons respectively.

In the 4-layer neural model, the hidden layer had 4 layers with 6, 12, 12, 6 neurons respectively.

	R^2 score	RMSE	MAPE
2 layers + 200 epochs	-0.8025	300.9320	38.1465
3 layers + 200 epochs	0.2214	197.7833	26.9871
4 layers + 200 epochs	0.1777	203.2548	28.8034
2 layers + 300 epochs	-0.1150	236.6839	29.7857
3 layers + 300 epochs	0.2873	189.2215	24.3515
4 layers + 300 epochs	0.2197	197.9977	26.0832
2 layers + 400 epochs	-8.1909	679.5374	94.0803
3 layers + 400 epochs	0.3303	183.4266	25.2099
4 layers + 400 epochs	-8.1906	679.5247	94.0780

Correlation Matrix:



	slambx	slamby	time_taken	L1_cache	L2_cache	L3_cache	CPU
slambx	1	1.5e-16	0.56	1e-15	-1.5e-16	-1.7e-15	1.6e-15
slamby	1.5e-16	1	0.55	3.1e-16	3.1e-16	-4e-17	-1.3e-16
time_taken	0.56	0.55	1	-0.38	-0.43	0.068	-0.14
L1_cache	1e-15	3.1e-16	-0.38	1	0.69	-0.63	0.45
L2_cache	-1.5e-16	3.1e-16	-0.43	0.69	1	-0.13	0.23
L3_cache	-1.7e-15	-4e-17	0.068	-0.63	-0.13	1	-0.79
CPU	1.6e-15	-1.3e-16	-0.14	0.45	0.23	-0.79	1

7. FUTURE SCOPE:

- The parameter KELEC can also be taken into consideration as a parameter while performing the tests and generating results.
- This project uses the L1 cache, L2 cache, L3 cache, and the CPU of the system as features. But more properties like processor speed and disk I/O speed can be included.
- The serial code can be parallelized to utilize the power of parallel processing in cores and threads.
- A new version of the code which dynamically varies the area of interest based on the propagation of plasma has become available. This reduces the computation which needs to be carried out, and hence can be further analyzed.