**SRI SHANMUGHA COLLEGE OF ENGINEERING AND TECHNOLOGY**

**Department of Biomedical Engineering**

**SMART PARKING**

**Phase 4 Submission Document**

**Team members**

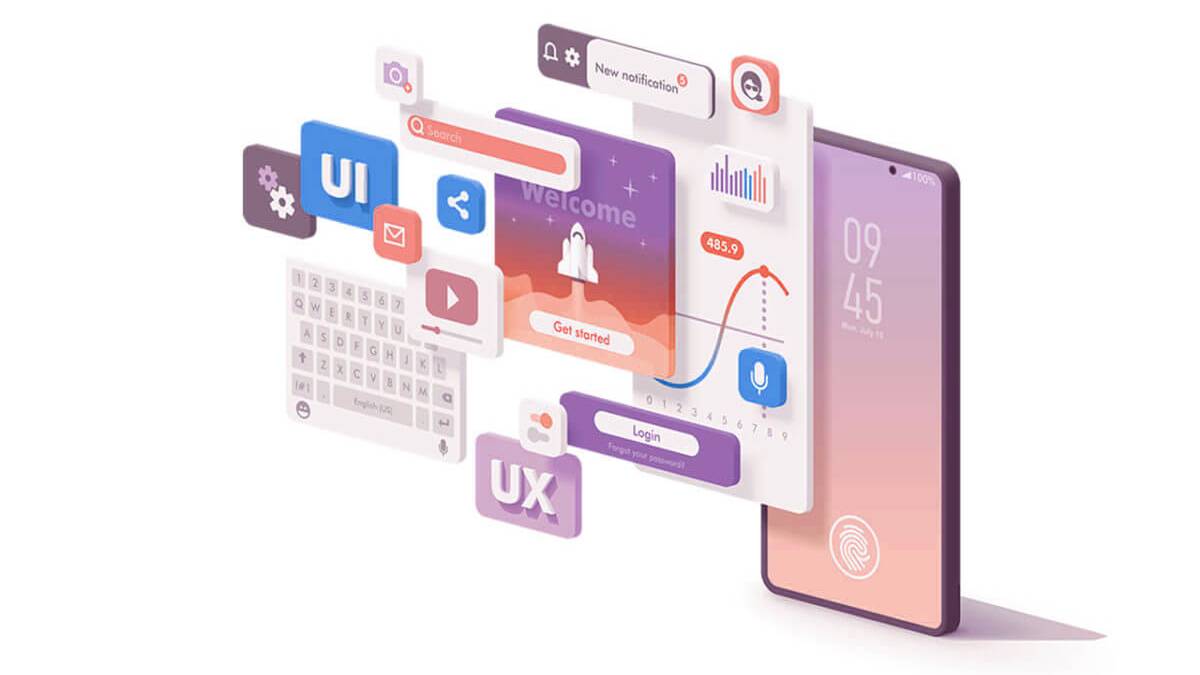
S. Swetha- (732721121055)  
 M. Sindhu - (732721121046)  
 T. Srimathi- (732721121051)  
 C. Chanthruba- (732721121007)

R. Bharathi- (732721121006)



**Introduction:**

The goal of parking system project is to reserve parking spot for a car/vehicle before it arrives. One of the most problems that the driver faces is finding a free parking spot, so many driver stopping their cars at the edges of the street. Therefore, we choose this to prevent the frustration of finding a parking spot and they can reserve a spot when they stay at home. It is an IOT based project.



**Overview of the process:**

Smart parking systems aim to improve the efficiency and convenience of parking by leveraging technology and data to assist drivers in finding available parking spaces. The process of a typical smart parking system involves several key components and steps:

**Sensor Deployment**: Sensors, such as ultrasonic sensors, infrared sensors, or cameras, are installed in parking spaces or areas to monitor their occupancy status. These sensors can detect whether a parking spot is vacant or occupied.

**Data Collection:** The deployed sensors continuously collect data on the availability of parking spaces. This data is sent to a central server or cloud-based platform for processing and analysis.

**Data Processing and Analysis**: The data collected from sensors is processed and analyzed in real-time to determine the availability of parking spaces. Machine learning algorithms may be used to predict parking spot availability based on historical data and real-time sensor readings.

**User Interface:** A user-friendly interface, often in the form of a mobile app or a website, is made available to drivers. This interface provides real-time information about parking availability, including the number of available spots and their locations.

**Navigation and Booking:** Drivers can use the app or website to search for available parking spots near their destination. The system provides navigation instructions to guide drivers to the selected parking spot. Some systems also allow users to reserve parking spaces in advance.

**Payment and Billing:** Payment for parking can be integrated into the app or website. Drivers can make payments electronically, reducing the need for physical tickets or cash. Parking fees can be automatically calculated based on the duration of the stay.

**Feedback and Alerts:** The system can provide alerts to drivers, such as reminders of when their parking time is about to expire. Drivers can also provide feedback on the parking experience, helping operators improve the system.

**Management and Analytics:** Operators of the smart parking system have access to a management dashboard where they can monitor the status of parking spaces, gather analytics, and make data-driven decisions to optimize parking operations.

**Integration:** Smart parking systems can be integrated with other urban infrastructure systems, such as traffic management and public transportation, to create a more holistic approach to urban mobility.

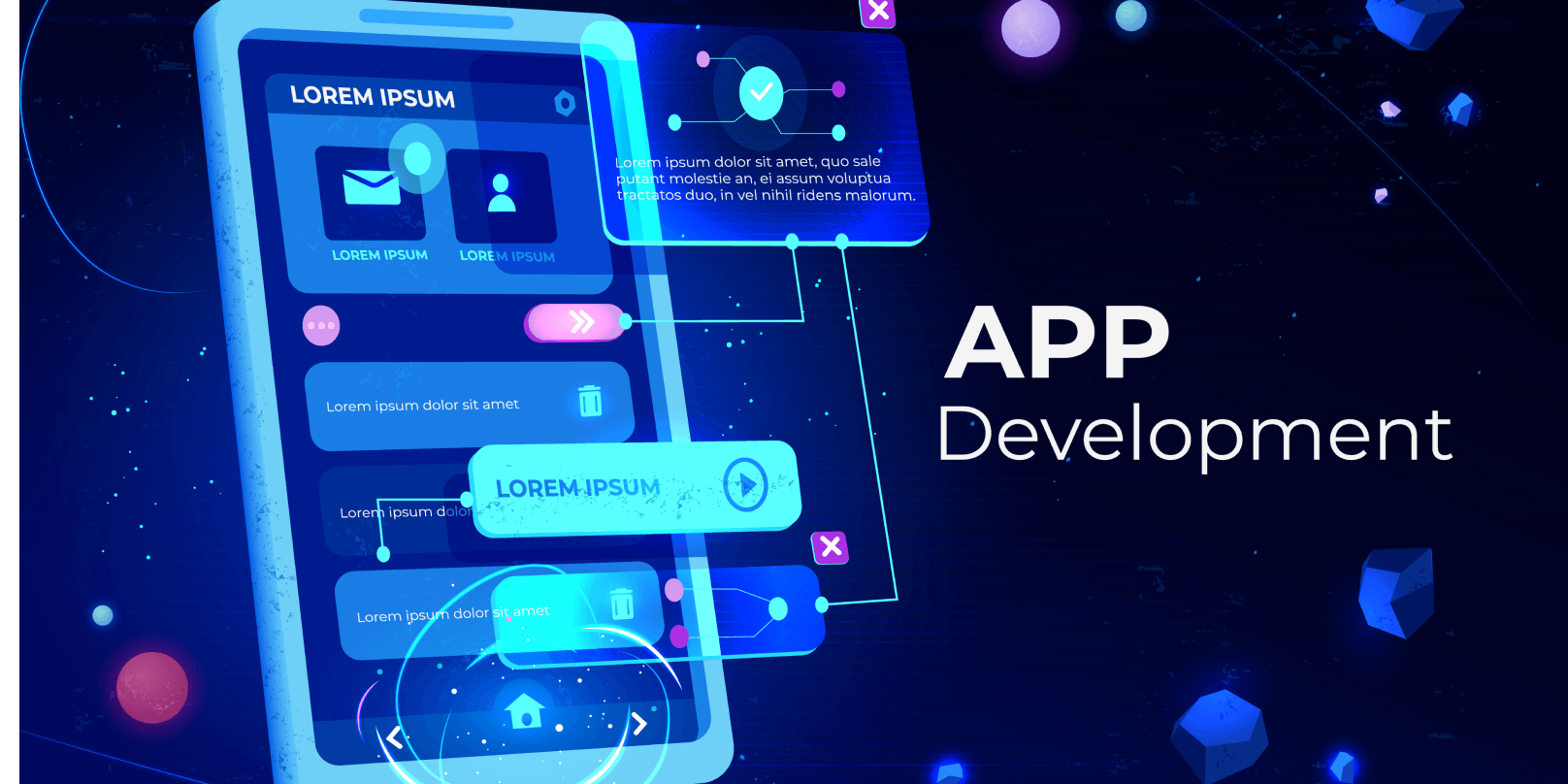
**Developing a mobile Application:**

Developing a mobile application involves several key steps, from planning and design to coding, testing, and deployment. Here is a general overview of the mobile app development process:

**1. Define Your App Idea:** Start by clearly defining your app's purpose, target audience, and the problems it aims to solve. Consider conducting market research to understand user needs and competition.

**2. Choose a Platform**: Decide whether you want to develop the app for iOS, Android, or both (cross-platform). Each platform has its development tools and languages (Swift/Objective-C for iOS, Java/Kotlin for Android), and you may also explore cross-platform frameworks like React Native or Flutter.

**3. Plan Your App:** Create a detailed project plan that outlines the app's features, functionalities, user interface, and user experience. Consider creating wireframes and mockups to visualize your app's design.



**4. Set Up the Development Environment:** Install the necessary software development kits (SDKs), IDEs (Integrated Development Environments), and tools for your chosen platform(s).

**5. Design the User Interface (UI):** Create the app's user interface design, ensuring it is user-friendly and visually appealing. Use design tools like Adobe XD, Sketch, or Figma.

**6. Develop Your App:** Write the code for your app, implementing the features and functionalities you've planned. Follow best coding practices and guidelines for your chosen platform.

**7. Test Your App:** Conduct thorough testing of your app to identify and fix bugs, usability issues, and performance problems. Use emulators/simulators and real devices for testing.

**8. Debug and Optimize:** Debug and optimize your app's code to ensure it runs smoothly, efficiently, and without crashes. Optimize for performance and memory management.

**9. Implement Backend Services (if needed):** If your app requires server-side functionality (e.g., user authentication, data storage, or real-time updates), develop the necessary backend components or use cloud-based services.

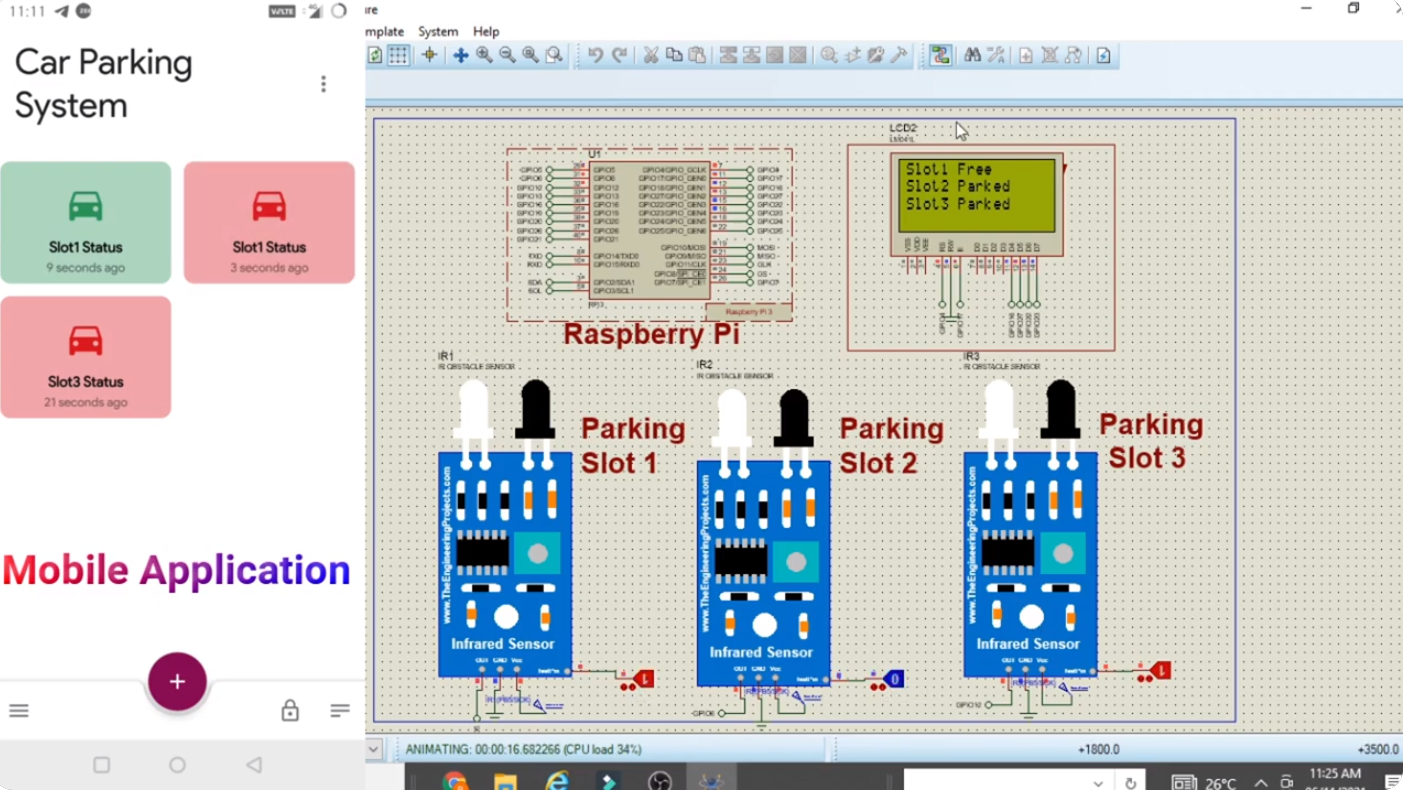
**10. Add App Features:** Integrate additional features like user authentication, social sharing, geolocation, push notifications, and any APIs or third-party services your app relies on.

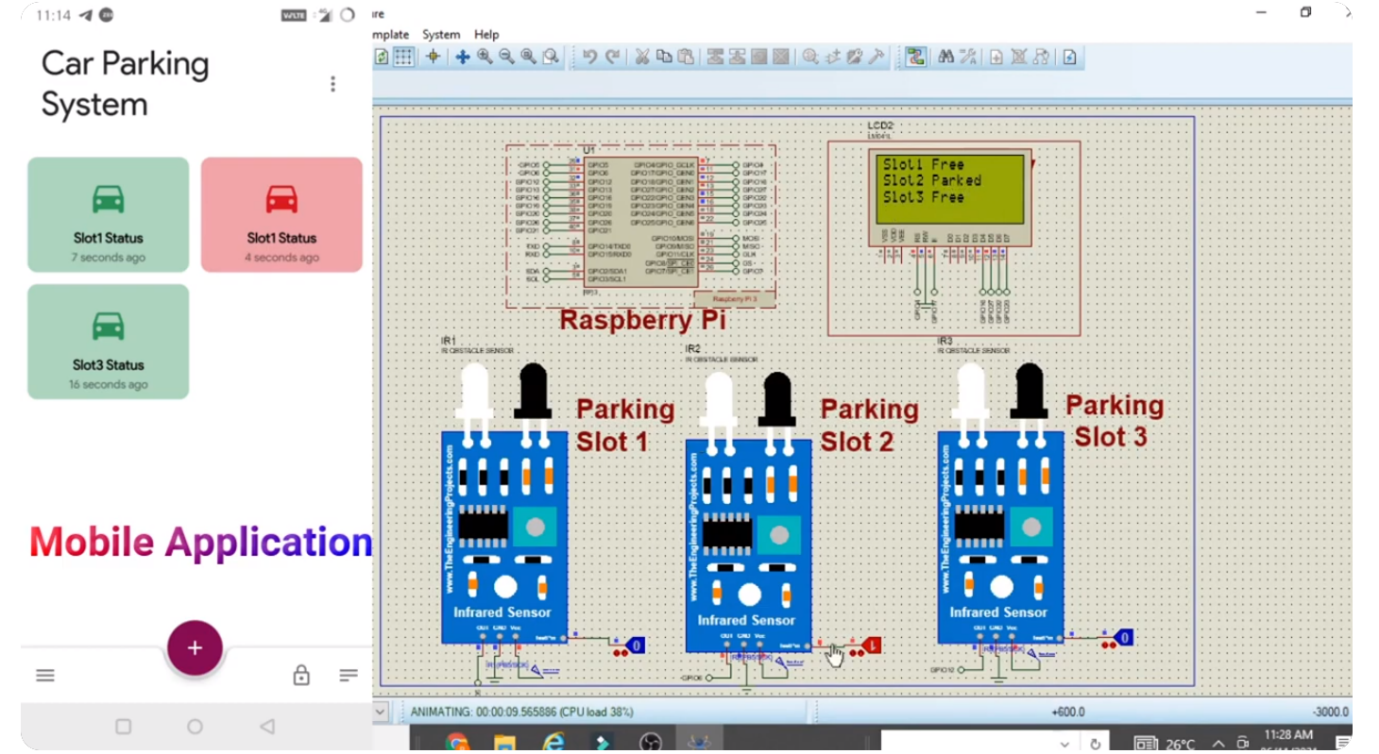
**11. Security and Privacy:** Implement security measures to protect user data and privacy. Secure data transmission, use encryption, and follow best practices for user authentication.

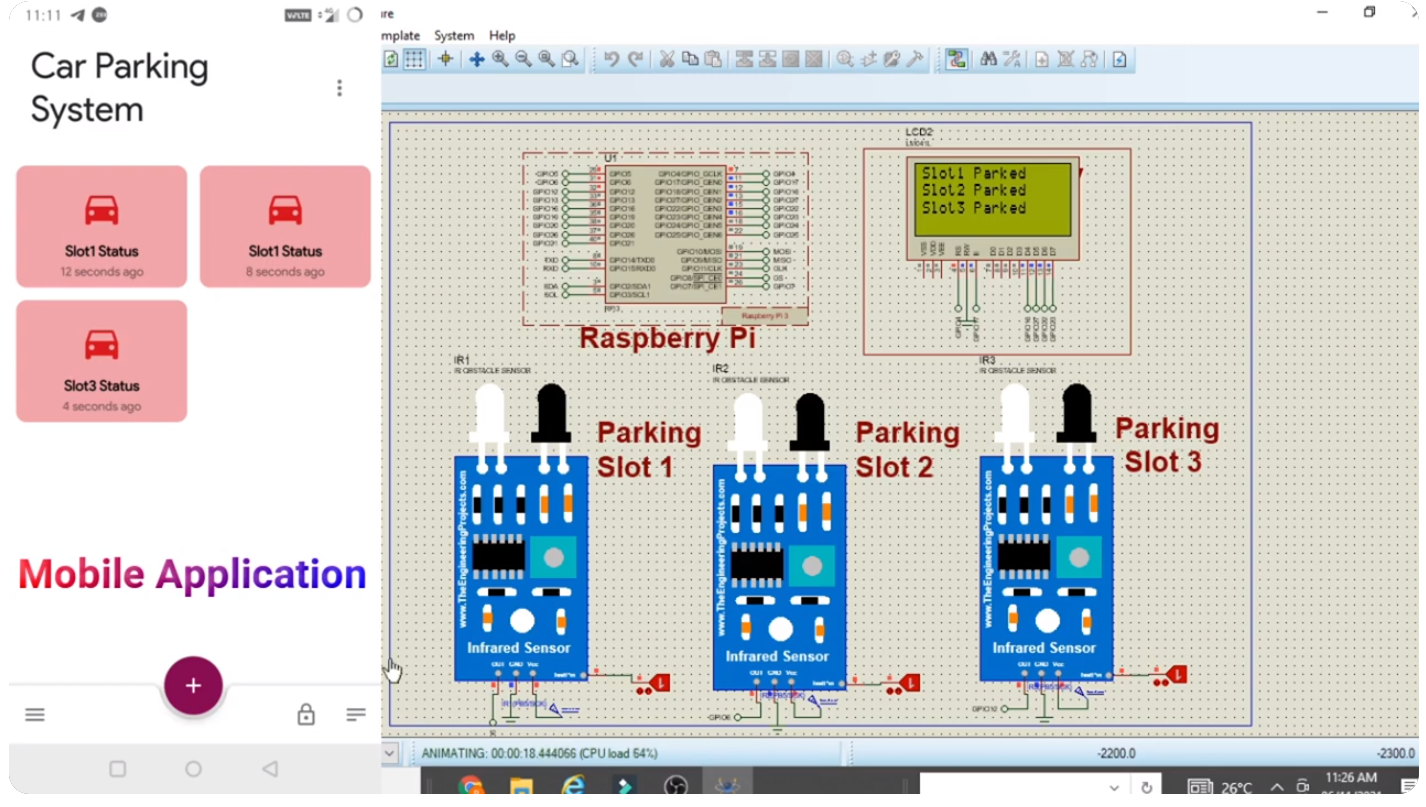
**12. Test on Real Devices:**  Test your app on various real devices to ensure compatibility and performance on a range of screen sizes and operating system versions.

**Execution of realtime parking availability:**

The execution of a parking availability system involves several key components and processes, including data collection, analysis, user interface development, and communication. “***It is a live parking status on mobile application using Proteus Simulation Software.”***







**Python scripts on Raspberry Pi:**

import RPi.GPIO as GPIO

import time

# Set GPIO pins

trigger\_pin = 18

echo\_pin = 17

# Set GPIO mode

GPIO.setmode(GPIO.BCM)

# Setup GPIO pins

GPIO.setup(trigger\_pin, GPIO.OUT)

GPIO.setup(echo\_pin, GPIO.IN)

try:

while True:

# Send a trigger pulse

GPIO.output(trigger\_pin, True)

time.sleep(0.00001)

GPIO.output(trigger\_pin, False)

pulse\_start = time.time()

while GPIO.input(echo\_pin) == 0:

pulse\_start = time.time()

pulse\_end = time.time()

while GPIO.input(echo\_pin) == 1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150 # Speed of sound = 343 m/s (17150 cm/s)

# Check if a car is present based on distance

if distance < 10: # Adjust this value based on your parking space layout

print("Parking space is occupied")

else:

print("Parking space is vacant")

time.sleep(1) # Check the status periodically

except KeyboardInterrupt:

GPIO.cleanup()

**Performance Evaluation Techniuqes:**

To evaluate the performance of deep LSTM network we used the root mean square error (RMSE), mean absolute error (MAE), mean squared error (MSE), median absolute error (MdAE), and mean squared log error (MSLE). The mathematical formulation of these performance evaluation techniques is defined as:

RMSE= √1/𝑛𝑛 ∑ (𝑦𝑦𝑖𝑖 − 𝑦𝑦^𝑖𝑖) 𝑛𝑛 2 𝑖𝑖=1 (1)

MAE = 1/𝑛𝑛 ∑ |𝑦𝑦𝑖𝑖 − 𝑦𝑦^ 𝑖𝑖 | 𝑛𝑛 𝑖𝑖=1 (2)

MSE =1/𝑛𝑛 ∑ (𝑦𝑦𝑖𝑖 − 𝑦𝑦^𝑖𝑖) 𝑛𝑛 2 𝑖𝑖=1 (3)

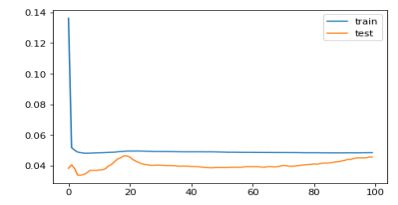
MdAE = median (∑ |𝑦𝑦𝑖𝑖 − 𝑦𝑦^ 𝑖𝑖 | 𝑛𝑛 𝑖𝑖=1 ) (4)

MSLE = 1/𝑛𝑛 ∑ log(𝑦𝑦𝑖𝑖) + 1 − log ( (𝑦𝑦^ 𝑖𝑖 ) + 1) 2 𝑛𝑛 𝑖𝑖=1 (5)

The symbol yi is the actual avail parking space that is computed by taking the difference in occupancy and capacity values. Whereas, yˆi is the predicted available parking space, predicted by deep LSTM network. The difference between actual available parking space and predicted parking space is computed by y y i i − ˆ . The experimental results achieved using a deep LSTM network.

**Model training:**

Model training in the context of the Internet of Things (IoT) involves training machine learning models to analyze data generated by IoT devices. IoT devices can produce vast amounts of data, and machine learning models can help extract valuable insights, make predictions, and enable automation based on this data.



**Data Collection:** IoT devices collect data, which can include sensor readings, images, audio, or other types of data. This data is typically stored and preprocessed before it's used for training.

**Data Preprocessing:** Raw IoT data often requires preprocessing to clean, format, and transform it into a suitable format for model training. Data preprocessing may involve handling missing values, normalizing data, and feature engineering.

**Data Labeling (if applicable):** In some IoT applications, data may need labeling, such as tagging anomalies in sensor data for predictive maintenance.

**Training Data Split:** The dataset is divided into training data, validation data, and test data to evaluate and fine-tune the model.

**Model Selection:** Choose an appropriate machine learning or deep learning model architecture based on the specific IoT application. Common models include decision trees, support vector machines, neural networks, and more.

**Feature Selection**: Identify relevant features or input variables that are important for making predictions. Feature selection can improve model efficiency and accuracy.

**Model Training:** The selected model is trained using the training data. During training, the model learns to make predictions or classifications based on the input data. Training may involve optimizing model parameters using optimization algorithms (e.g., gradient descent) to minimize a defined loss function.

**Hyperparameter Tuning:** Adjust hyperparameters like learning rate, batch size, and the number of layers or neurons in the model to optimize model performance. Hyperparameter tuning is often done using techniques like grid search or random search.

**Validation and Testing:** The trained model is evaluated using the validation dataset to assess its performance and generalization capability. Additional testing is performed on the test dataset to confirm the model's ability to make predictions on unseen data.

**Continuous Monitoring:** IoT models may need ongoing monitoring to ensure they continue to perform well over time. Drift detection mechanisms can be implemented to detect changes in data patterns that might require model retraining.

**Model evaluation:**

Model evaluation is the process of assessing the performance of a machine learning model on unseen data. This is important to ensure that the model will generalize well to new data. There are a number of different metrics that can be used to evaluate the performance of a house price prediction model. Some of the most common metrics include:

**Mean squared error (MSE**): This metric measures the average squared difference between the predicted and actual house prices.

**Root mean squared error (RMSE):** This metric is the square root of the MSE.

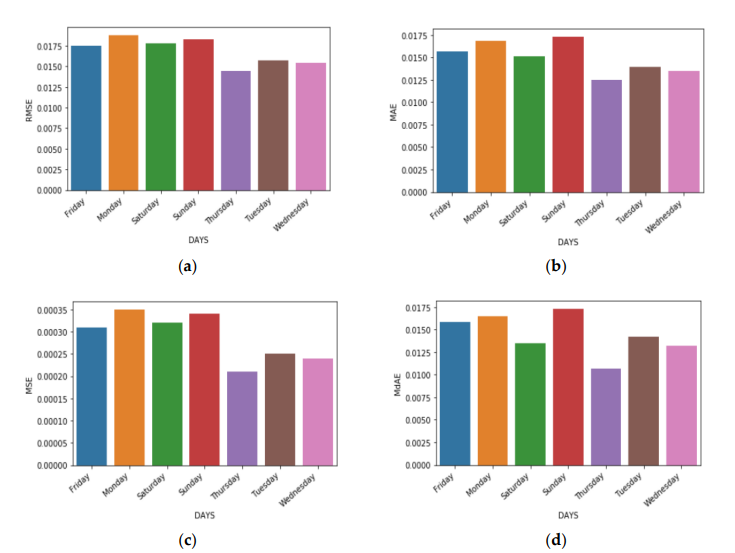
**Mean absolute error (MAE):** This metric measures the average absolute difference between the predicted and actual house prices.

**R-squared**: This metric measures how well the model explains the variation in the actual house prices. In addition to these metrics, it is also important to consider the following factors when evaluating a house price prediction model:

**Bias:** Bias is the tendency of a model to consistently over- or underestimate house prices.

**Variance:** Variance is the measure of how much the predictions of a model vary around the true house prices.

**Interpretability:** Interpretability is the ability to understand how the model makes its predictions. This is important for house price prediction models, as it allows users to understand the factors that influence the predicted house prices.



1. Calculate the evaluation metrics. There are a number of different

evaluation metrics that can be used to assess the performance of a

machine learning model, such as R-squared, mean squared error

(MSE), and root mean squared error (RMSE).

1. Interpret the evaluation metrics. The evaluation metrics will

give you an idea of how well the model is performing on unseen data. If

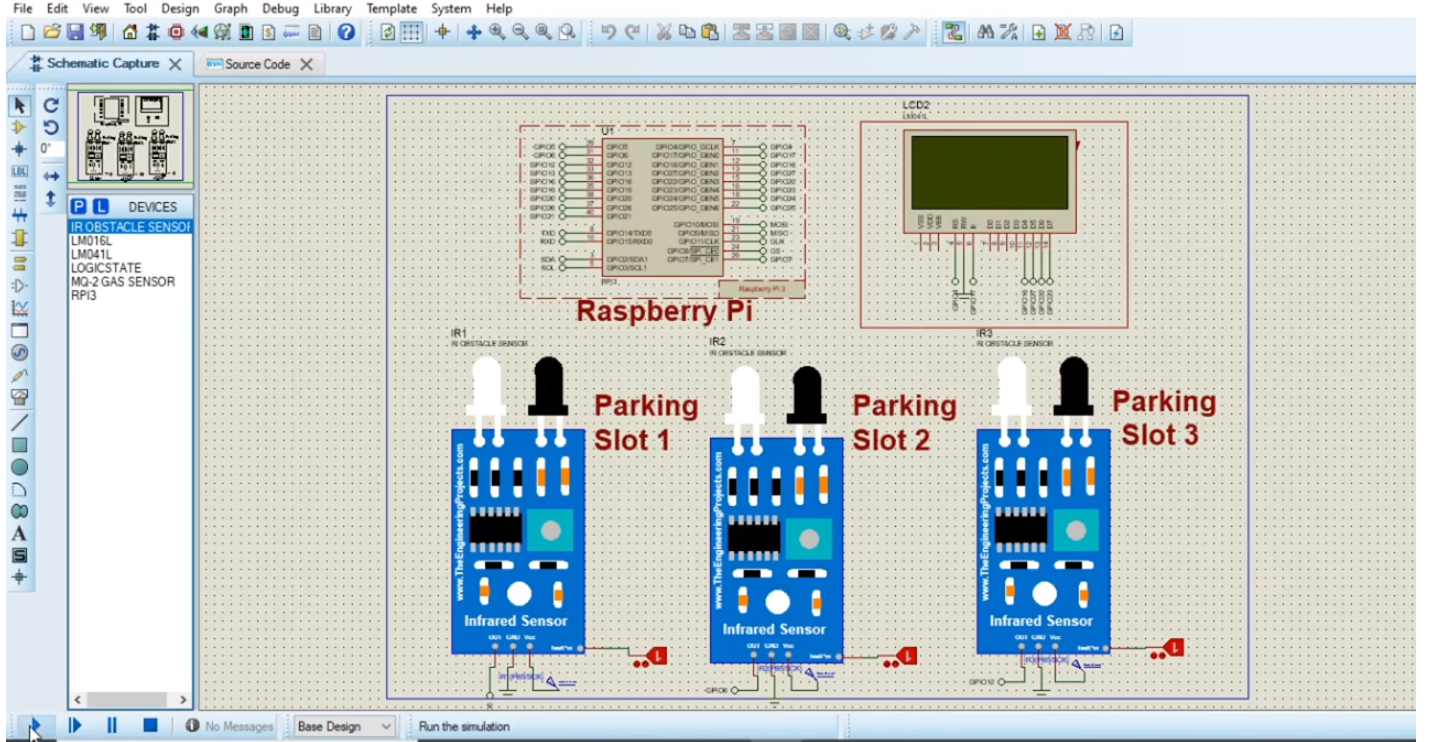
the model is performing well, then you can be confident that it will

generalize well to new data. However, if the model is performing poorly,

then you may need to try a different model or retune the

hyperparameters of the current model.

**System Output:**





**Conclusion:**

Smart parking solutions offer a comprehensive approach to tackling the challenges associated with parking in today's urban environments. While initial implementation can be an investment, the long-term benefits in terms of reduced traffic congestion, environmental impact, and improved user experiences make smart parking a valuable addition to modern cities. As technology continues to advance, smart parking systems are likely to evolve and become even more integral to urban planning and sustainability efforts.