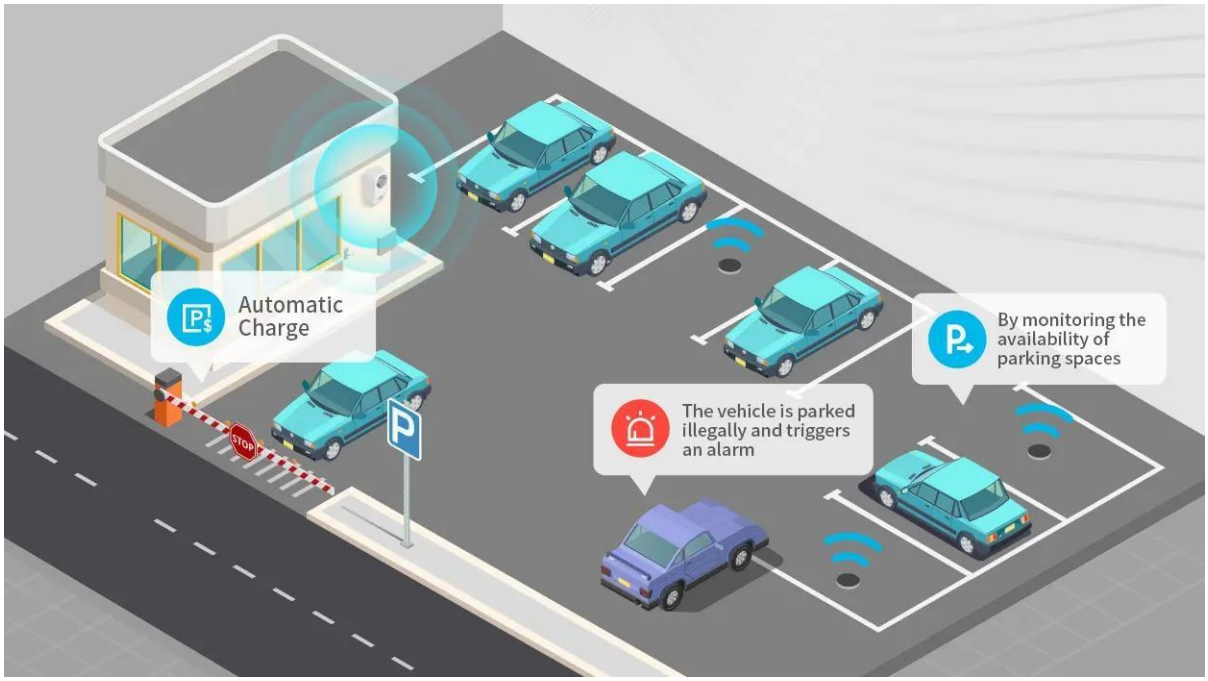
Smart Parking

Phase 5 Submission Document

Project: Smart Parking

Phase 4:



Development part 1:

Clearly outline the problem statement, design thinking process, and the phases of development.

Problem Statement:

- Inefficient use of parking spaces: Traditional parking systems often lead to inefficient use of parking spaces, as drivers struggle to find available spots, resulting in congestion, wasted time, and increased pollution.
- Lack of real-time information: Drivers lack accurate, real-time information about parking availability, leading to frustration and wasted time searching for parking.

- High maintenance and management costs: Traditional parking facilities require manual management, which is costly and often leads to revenue loss due to human error and theft.

Impact: Explain the consequences or negative effects of the problem.

The negative effects of traditional parking management include traffic congestion, environmental pollution, wasted time and stress for drivers, inefficient space utilization, safety concerns, inconsistent pricing, revenue loss, and user frustration. In addition, these problems can lead to inefficiencies in urban planning, limited accessibility for certain demographics, negative environmental impact, and a significant allocation of urban land to parking. A smart parking system aims to mitigate these consequences by improving traffic flow, reducing emissions, enhancing user experiences, optimizing space utilization, and providing valuable data for urban planning, ultimately leading to more efficient and sustainable urban environments.

Scope: Define the boundaries of the problem to avoid ambiguity.

The scope of the problem is defined as the challenges and inefficiencies associated with traditional parking management, encompassing issues such as inefficient space utilization, traffic congestion, environmental pollution, revenue loss, user frustration, and safety concerns. It includes the negative impacts on both drivers and urban environments, as well as the limitations of existing parking systems in terms of equity and accessibility. The scope of this problem extends to the need for a comprehensive solution that improves the overall parking experience, reduces environmental harm, optimizes resource allocation, and enhances urban planning efficiency through the implementation of a smart parking system.

Stages of the Design Thinking Process:

1. **Empathize:** This stage involves understanding the problem from the user's perspective. Design thinkers seek to empathize with the end users to gain deep insights into their needs, challenges, and emotions. Methods such as interviews, observations, and surveys are used to gather qualitative data about the user's experiences and pain points.

2. **Define:** In this phase, the insights gathered in the empathize stage are synthesized to define a clear and specific problem statement. This step is about framing the problem in a way that sets the direction for the rest of the process. It involves identifying the root causes of the problem and understanding its context.

3. **Ideate:** Ideation is a creative phase where cross-functional teams brainstorm and generate a wide range of potential solutions to the defined problem. The emphasis is on quantity and diversity of ideas rather than evaluating them. Techniques like brainstorming, mind mapping, and sketching are commonly used.

4. **Prototype:** In this stage, design thinkers create low-fidelity representations of their ideas, which can be physical or digital prototypes. These prototypes are used to quickly and cheaply test concepts and gather feedback. Prototyping helps in refining and visualizing ideas.

5. **Test:** The testing phase involves presenting the prototypes to end users for feedback and evaluation. The feedback is used to refine the solutions iteratively. This stage often leads to further insights, and the process may cycle back to earlier stages if needed.

6. **Implement (or Scale):** Once a viable solution is developed and validated through testing, it can be implemented or scaled for real-world use. Implementation involves bringing the solution to market, while scaling involves expanding the impact of the solution to a larger audience or context**.**

Phases of Development:

Definition:

Development phases: planning, design, testing, deployment, maintenance, evaluation. Iterative process, meeting requirements and user needs.

Typical Phases of Development:

Conceptualization/Planning: Define the goals, scope, and initial concept of the **solution.**

Design: Create detailed plans, blueprints, or specifications for the solution.

Development/Implementation: Build the solution based on the design.

Testing/QA: Assess the solution for functionality, reliability, and user satisfaction.

Deployment/Release: Introduce the solution to the intended users or audience.

Maintenance/Support: Provide ongoing support, updates, and improvements.

Example:

In a software development project, the phases might look like this:

Conceptualization: Define the software requirements and plan the project.

Design: Develop the software architecture, user interface, and database structure.

Development: Write the actual code based on the design specifications.

Testing: Identify and fix bugs, ensure the software meets quality standards.

Deployment: Release the software to users.

Maintenance: Address issues, release updates, and provide ongoing support.

Choosing a classification algorithm and designing an effective model training process are critical steps in developing a machine learning model. The choice of the algorithm depends on the nature of the problem, the characteristics of the data, and the desired outcome. Below is an explanation of the factors influencing the selection of a classification algorithm and an outline of the model training process.

Choice of Classification Algorithm:

Binary or Multiclass Classification: Determine whether the problem involves classifying data into two classes or more than two classes.

Imbalance: If the classes are imbalanced, where one class has significantly fewer samples, algorithms robust to imbalanced data (e.g., ensemble methods) may be preferred.

Data Characteristics:

Linear Separability: If the data is linearly separable, linear models like Logistic Regression or Support Vector Machines may be effective.

Non-linearity: For complex, non-linear relationships in the data, algorithms like Decision Trees, Random Forests, or Gradient Boosting methods might be suitable.

Data Size and Dimensionality:

Large Datasets: Deep learning models, such as neural networks, may be suitable for large datasets with high dimensionality.

Small Datasets: Simple models like Naive Bayes or k-Nearest Neighbors might work well with small datasets.

Interpretability:

Interpretability Requirement: Consider whether model interpretability is crucial. Decision Trees and Logistic Regression are generally more interpretable than complex models like neural networks.

Computational Resources:

Computational Complexity: Choose an algorithm that aligns with available computational resources. For example, deep learning models may require powerful GPUs.

Model Training Process: Data Preprocessing:

Cleaning: Handle missing values, outliers, and any noise in the dataset.

Feature Engineering: Create new features or transform existing ones to enhance the model's performance.

Normalization/Scaling: Standardize or normalize numerical features to ensure they are on a similar scale.

Data Splitting:

Training Set: The portion of data used for training the model.

Validation Set: A subset used to tune hyperparameters and avoid overfitting.

Test Set: Unseen data used to evaluate the model's performance.

Feature Selection:

Identify Relevant Features: Select features that contribute most to the model's predictive power.

Dimensionality Reduction: Use techniques like Principal Component Analysis (PCA) to reduce the number of features.

Model Selection:

Choose Algorithm: Based on the characteristics of the problem and data, select an appropriate classification algorithm.

Hyperparameter Tuning:

Grid Search or Random Search: Explore combinations of hyperparameter values to find the best-performing model.

Cross-Validation: Use techniques like k-fold cross-validation to assess model performance across different subsets of the training data.

Model Training:

Fit the Model: Train the model on the training dataset using the selected algorithm and optimal hyperparameters.

Validation: Validate the model on the validation set to ensure it generalizes well to unseen data.

Evaluation:

Test Set Evaluation: Assess the model's performance on the test set to estimate its real-world performance.

Metrics: Choose appropriate evaluation metrics based on the nature of the problem (**accuracy, precision, recall, F1-score, etc.).**

Iterative Refinement:

Feedback Loop: Based on the evaluation results, refine the model by adjusting hyperparameters, re-evaluating, and iterating as needed.

Deployment:

Deploy the Model: Once satisfied with the model's performance, deploy it for making predictions on new, unseen data.

Coding:

```
#define ECHO_PIN1 15 //Pins for Sensor 1
#define TRIG_PIN1 2 //Pins for Sensor 1

#define ECHO_PIN2 5 //Pins for Sensor 2
#define TRIG_PIN2 18 //Pins for Sensor 2

#define ECHO_PIN3 26 //Pins for Sensor 3
#define TRIG_PIN3 27 //Pins for Sensor 3
```

```
int LEDPIN1 = 13;
int LEDPIN2 = 12;
int LEDPIN3 = 14;

void setup() {
  Serial.begin(115200);
  pinMode(LEDPIN1, OUTPUT);
  pinMode(TRIG_PIN1, OUTPUT);
  pinMode(ECHO_PIN1, INPUT);

   pinMode(LEDPIN2, OUTPUT);
  pinMode(TRIG_PIN2, OUTPUT);
  pinMode(ECHO_PIN2, INPUT);

   pinMode(LEDPIN3, OUTPUT);
  pinMode(TRIG_PIN3, OUTPUT);
  pinMode(ECHO_PIN3, INPUT);
}

float readDistance1CM() {
  digitalWrite(TRIG_PIN1, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN1, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN1, LOW);
  int duration = pulseIn(ECHO_PIN1, HIGH);
  return duration * 0.034 /2 ;
}

float readDistance2CM() {
  digitalWrite(TRIG_PIN2, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN2, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN2, LOW);
  int duration = pulseIn(ECHO_PIN2, HIGH);
  return duration * 0.034 / 2;
}

float readDistance3CM() {
  digitalWrite(TRIG_PIN3, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN3, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN3, LOW);
  int duration = pulseIn(ECHO_PIN3, HIGH);
  return duration * 0.034 / 2;
```

```
}

void loop() {
  float distance1 = readDistance1CM();
  float distance2 = readDistance2CM();
  float distance3 = readDistance3CM();

  bool isNearby1 = distance1 > 200;
  digitalWrite(LEDPIN1, isNearby1);


  bool isNearby2 = distance2 > 200;
  digitalWrite(LEDPIN2, isNearby2);


  bool isNearby3 = distance3 > 200;
  digitalWrite(LEDPIN3, isNearby3);

  Serial.print("Measured distance: ");
  Serial.println(readDistance1CM());
  Serial.println(readDistance2CM());
  Serial.println(readDistance3CM());
  delay(100);
}
```