1.Genetic algorithm packman game:

**Project Overview**

In this project, I applied Genetic Algorithms (GA) to develop an AI agent capable of playing the classic Pac-Man game more effectively. The goal was to optimize the agent's performance by evolving a set of parameters that govern its decision-making process, ultimately enhancing its ability to navigate the game, avoid ghosts, and consume food.

**Project Details**

**1. Genetic Algorithm Application**

The core of this project involves using a Genetic Algorithm to fine-tune the behaviour of the Pac-Man agent. Genetic Algorithms are a class of optimization algorithms inspired by the principles of natural selection and genetics. In this project, the GA was employed to evolve a population of potential solutions (chromosomes), where each chromosome represents a different set of parameters for the Pac-Man agent.

**2. Agent Design**

The Pac-Man agent's behaviour is governed by a set of parameters encoded in a chromosome. Each chromosome is an array of values that determine the agent's strategies for various aspects of the game:

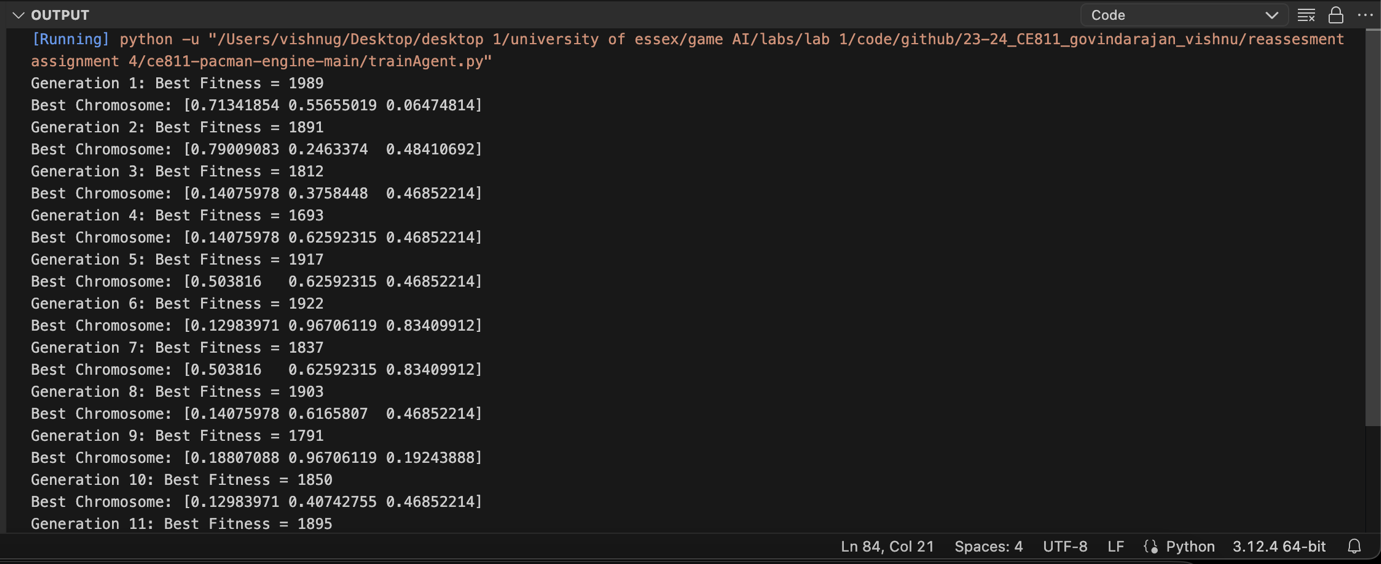
* Avoiding dangerous ghosts
* Seeking and consuming food
* Targeting scared ghosts when they are vulnerable

The agent's effectiveness is evaluated based on its game score, which serves as the fitness function in the GA.

**3. Evolution Process**

The evolutionary process consists of several key steps:

* **Initialization**: A population of chromosomes is randomly generated.
* **Evaluation**: Each chromosome is tested by running the Pac-Man game, and its performance is assessed based on the final score achieved.
* **Selection**: The best-performing chromosomes are selected based on their fitness scores.
* **Crossover**: Pairs of chromosomes are combined to produce offspring, combining traits from both parents.
* **Mutation**: Random changes are introduced to offspring chromosomes to maintain genetic diversity.

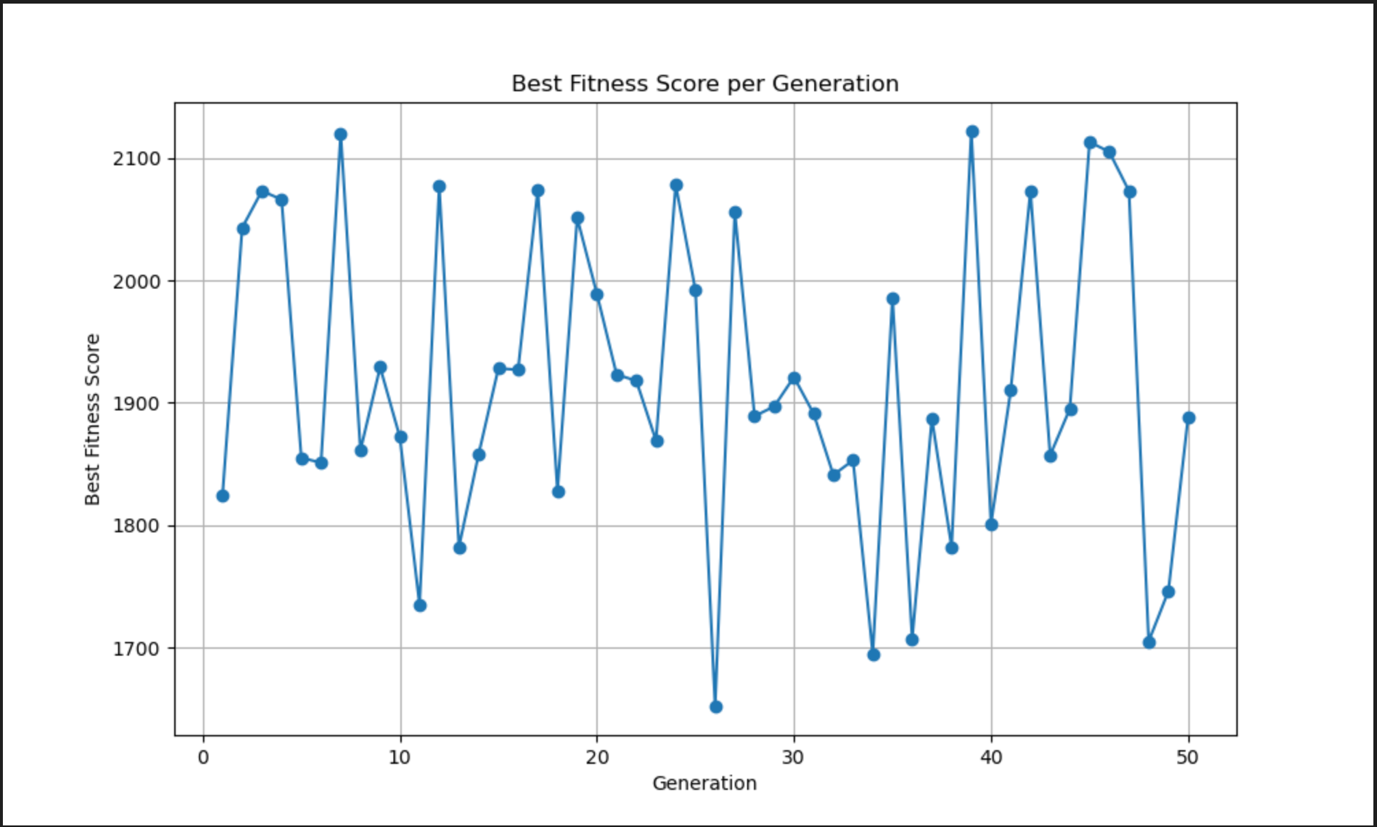


**4. Results**

After running the genetic algorithm, the agent’s performance improved as it adapted to better handle various game scenarios. The final optimized chromosome values were hardcoded into the agent for practical use.  
  
  
  
(put the video)

**5. Visualization**

The project includes a visual representation of the GA's progress over generations. A plot was generated to show how the best fitness score evolved, demonstrating the improvement in the agent's performance over time.



**Technical Stack**

* **Programming Languages**: Python
* **Libraries**: NumPy, Matplotlib
* **Game Framework**: Pac-Man Game Framework (CE811)
* **Optimization Algorithm**: Genetic Algorithm

**Conclusion**

This project showcases the application of Genetic Algorithms to enhance game-playing AI, demonstrating how evolutionary techniques can be used to optimize complex decision-making processes. By integrating GA with the Pac-Man game, the project illustrates how intelligent agents can be developed and refined to achieve better performance through adaptive learning.

#### A Rule-Based Pacman Agent with Dijkstra’s Shortest Path Algorithm for Ghost Avoidance and Food Collection

#### Project Overview

In this project, I developed an advanced Pac-Man AI agent utilizing Dijkstra’s Algorithm for pathfinding. This agent is designed to make intelligent decisions based on the layout of the maze, incorporating a sophisticated evaluation function to handle various game scenarios more effectively.

**Project Details**

**1. Pathfinding with Dijkstra’s Algorithm**

The key feature of this project is the implementation of Dijkstra’s Algorithm to enhance the Pac-Man agent’s navigation capabilities. Dijkstra’s Algorithm is a classic algorithm used for finding the shortest paths between nodes in a graph. By applying this algorithm to the Pac-Man game’s maze layout, the agent can calculate optimal paths to avoid ghosts and target food pellets.

**2. Implementation Highlights**

* **Pathfinding Functions**:
  + calculate\_neighbouring\_nodes: Computes neighbouring nodes for a given position, considering walls and layout boundaries.
  + calculate\_gscores: Computes the shortest path distances from a start node to all other nodes in the maze using Dijkstra’s Algorithm.
  + calc\_path\_A\_to\_B and calc\_path\_to\_point: Determine the optimal path between two points in the maze, translating this into a sequence of Pac-Man movements.

**3. Agent Design**

The agent is designed to make decisions based on a detailed evaluation of the game state, using both the pathfinding capabilities and a custom evaluation function:

* **getAction Method**: Chooses the best action from legal moves based on the evaluation of potential successor states.
* **evaluateBoardState Method**: Incorporates various factors such as distances to food, ghosts, and scared ghosts, to determine the best move. The evaluation function uses Dijkstra’s Algorithm's results to calculate distances, offering a more accurate assessment of the maze layout.

**4. Evaluation Function**

The evaluation function is designed to make the agent consider:

* **Distance to Nearest Food**: Rewards the agent for being closer to food pellets.
* **Proximity to Ghosts**: Penalizes the agent for being close to dangerous ghosts and rewards it for targeting scared ghosts.
* **Remaining Food Pellets**: Provides additional incentives based on the number of remaining food pellets.

**5. Results**

This approach ensures that the Pac-Man agent can navigate the maze more intelligently by considering both immediate and strategic factors. The agent's performance was evaluated through gameplay, demonstrating improved decision-making capabilities in various maze scenarios.

[upload video]

**Conclusion**

This project demonstrates the application of pathfinding algorithms in game AI, highlighting how advanced techniques can enhance decision-making and navigation in complex environments. By integrating Dijkstra’s Algorithm into the Pac-Man AI, the project showcases a significant advancement in creating intelligent and strategic game agents.

Neural network car driven using genetic algorithm

Welcome to my **Neural-Network Driving Game**! This project was created using **Python**, featuring a racing game where a car learns to drive on a track using a custom-built **neural network** and a **genetic algorithm**. Unlike many other projects that rely on machine learning libraries, I implemented the neural network from scratch using **NumPy**, which gave me deeper control over how the network operates and evolves.

**Key Features:**

* **Genetic Algorithm-Driven Learning**: The core innovation in this game is the use of a **genetic algorithm** (GA) to train the car's neural network to navigate the track. The GA simulates the process of natural selection, gradually evolving better-performing neural network "chromosomes" (sets of weights and biases) over hundreds of generations.
* **Hard-Coded Neural Network**: Instead of using neural network libraries like TensorFlow or PyTorch, I manually built the neural network using simple matrix operations with NumPy. This network has one hidden layer with a custom number of hidden nodes, and it uses **tanh activation functions** to produce decisions for steering and acceleration.
* **Three Driving Agents**:
  + **Keyboard Agent**: This agent allows manual driving using keyboard inputs.
  + **Autonomous Agent**: This uses a basic hand-coded algorithm to make decisions based on the car's position on the track.
  + **Neural Agent**: The neural agent is controlled by the neural network, which improves over time through genetic algorithm training.

**How It Works:**

The game consists of a car trying to drive around a racetrack without going off course. The track is randomly generated, with the car starting at a designated point and using steering and speed controls to stay on track.

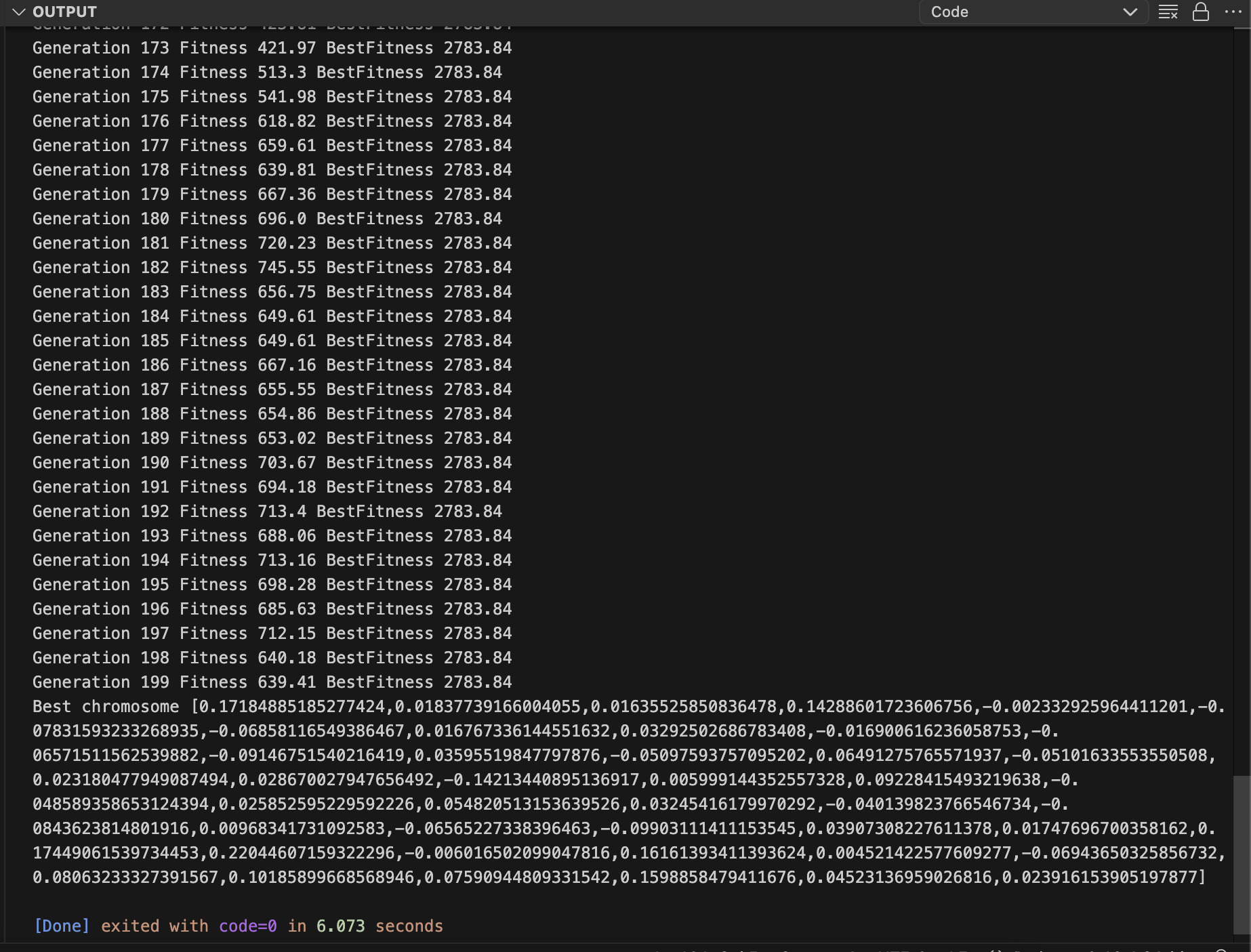
1. **Neural Network Structure**: The neural network takes in four inputs:
   * The car’s position relative to the track centre (cross-track error or XTE).
   * The road's orientation relative to the car’s current direction.
   * The distance travelled around the track.
   * Additional information like trigonometric functions of the distance.

It then processes this information through a hidden layer and outputs two values:

* + **Steering Control**: Whether to turn left, right, or go straight.
  + **Speed Control**: How fast the car should move (accelerate or decelerate).

[upload video]

1. **Genetic Algorithm**: The genetic algorithm generates random initial chromosomes (neural network weights and biases), which are tested over multiple simulations to evaluate their fitness (distance travelled on the track). The best-performing chromosomes are then mutated and evolved over generations to create a more optimal driving behaviour. This process continues until the neural network learns to drive the car effectively.



**Custom Features:**

* **Track Layout**: The track can have multiple curves and straight sections, adding complexity to the car's driving challenges.
* **Collision Detection**: The car must stay within the bounds of the track or it will crash, prompting a reset.
* **Graphical Display**: The game uses **Pygame** for graphical rendering, displaying the track and the car's movement in real time.

**Why This Project is Unique:**

* **No External Libraries for Neural Networks**: Unlike most machine learning projects, this project implements neural networks entirely from scratch using NumPy, providing a deeper understanding of the underlying mathematics.
* **Evolutionary Learning**: The use of a genetic algorithm adds an exciting twist to traditional neural network training. Instead of backpropagation, the network evolves over time, learning to improve through mutation and selection, similar to biological evolution.
* **Real-Time Visualization**: You can watch the learning process as the car navigates the track, improving its performance with each generation.

**Technologies Used:**

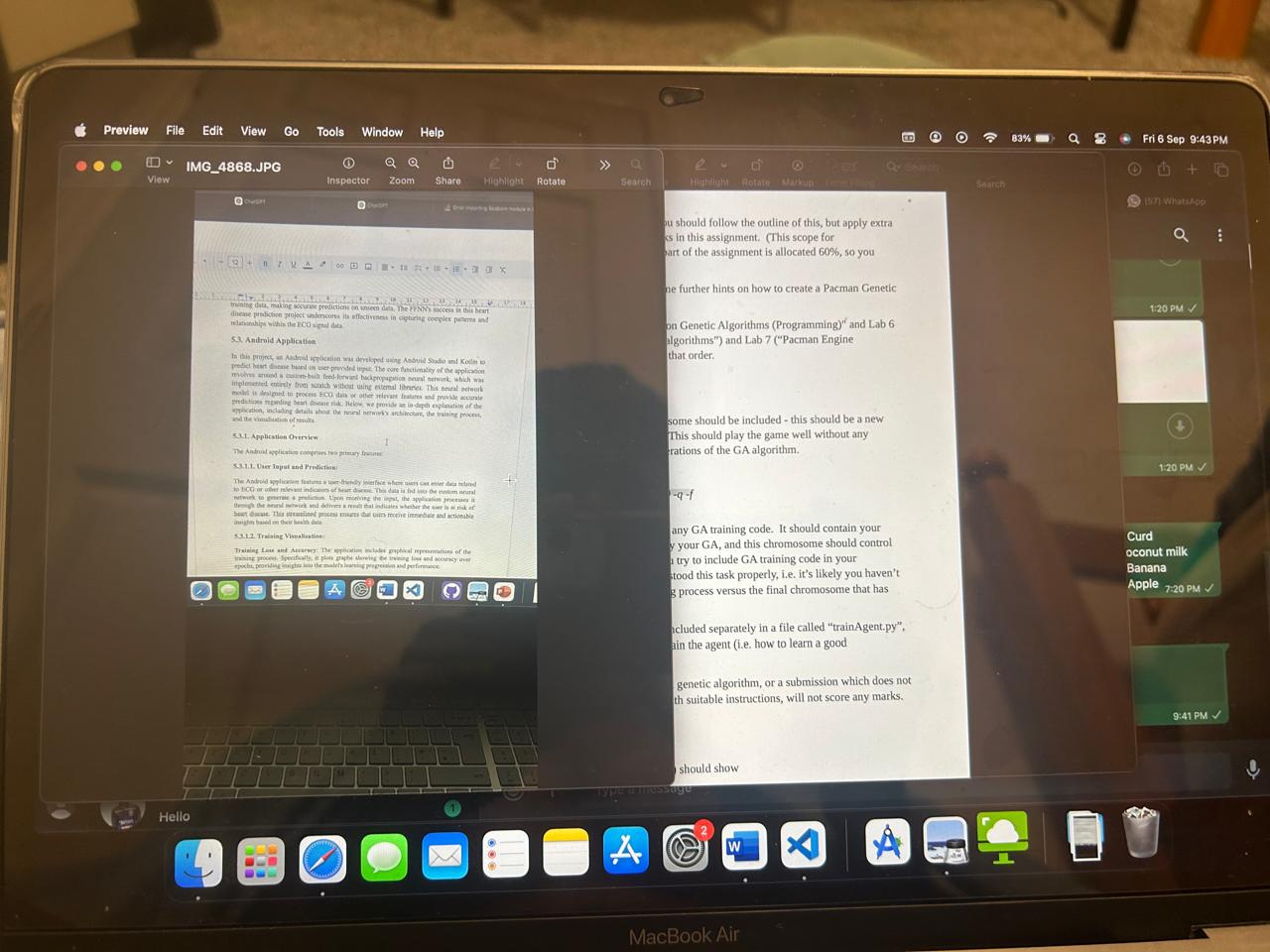
* **Python**: The core programming language used for the game logic and neural network.
* **NumPy**: Used for matrix and vector operations in the neural network.
* **Pygame**: A popular Python library for game development, responsible for rendering the graphics of the car and track.
* **Genetic Algorithm**: Implemented to evolve the neural network's weights and biases, optimizing the car's performance over time.

With this project, I explored the fascinating intersection of artificial intelligence, game development, and evolutionary computation. It serves as both a fun game and a demonstration of how neural networks can learn through genetic evolution, paving the way for further explorations in AI-driven simulations.

Robotics:

Right edge following behaviour using PID controller

In this project, I developed a **Right Edge Following Robot** using ROS (Robot Operating System) and a **PID (Proportional-Integral-Derivative)** controller to maintain a safe distance from obstacles while navigating along the right edge of walls or objects. This project is designed to integrate real-time sensor data with intelligent movement, making the robot capable of smooth and efficient obstacle avoidance.



[upload video]

#### Key Features:

1. **PID Controller Implementation**:
   * The robot uses a PID controller to adjust its movement, allowing precise and adaptive control. The PID controller continuously calculates an error value as the difference between the current distance from the wall (measured by LIDAR sensors) and a desired distance. It then applies corrections to the robot’s angular velocity to maintain a stable course.
   * The PID controller uses three parameters:
     + **Proportional (P)**: Determines the reaction to the current error.
     + **Integral (I)**: Accounts for the accumulation of past errors.
     + **Derivative (D)**: Predicts future error based on the rate of change.
2. **LIDAR Sensor Data Integration**:
   * The robot employs LIDAR sensors to continuously scan the environment and retrieve distance measurements from the surroundings. The regions scanned include the **front**, **right**, and **left** of the robot.
   * The robot focuses on the **right edge** for guidance, ensuring it stays at a safe, predefined distance from obstacles.

#### Code Breakdown:

* **LaserScan Callback**: The LIDAR sensor readings are processed in the clbk\_laser() function. It divides the environment into different regions such as **front**, **right**, **front-right**, **front-left**, and **left**, providing the robot with spatial awareness.
* **Movement Function**: This is the core of the robot’s behaviour. Using the PID controller, it adjusts the robot’s angular velocity (msg.angular.z) based on the difference between the current and desired distances from the right wall or obstacle. It keeps the robot moving forward at a controlled pace (msg.linear.x).
* **Timer Callback**: A timer is used to periodically publish the calculated velocity commands, ensuring that the robot moves continuously based on the latest sensor readings.

#### How It Works:

1. The robot starts moving forward.
2. The LIDAR sensors detect obstacles, and the robot focuses on maintaining a specified distance from the right wall or object.
3. If the robot gets too close to the wall, the **PID controller** adjusts its course, steering it away.
4. The robot continuously moves forward, making slight adjustments to its path to stay aligned with the right edge.

#### Conclusion:

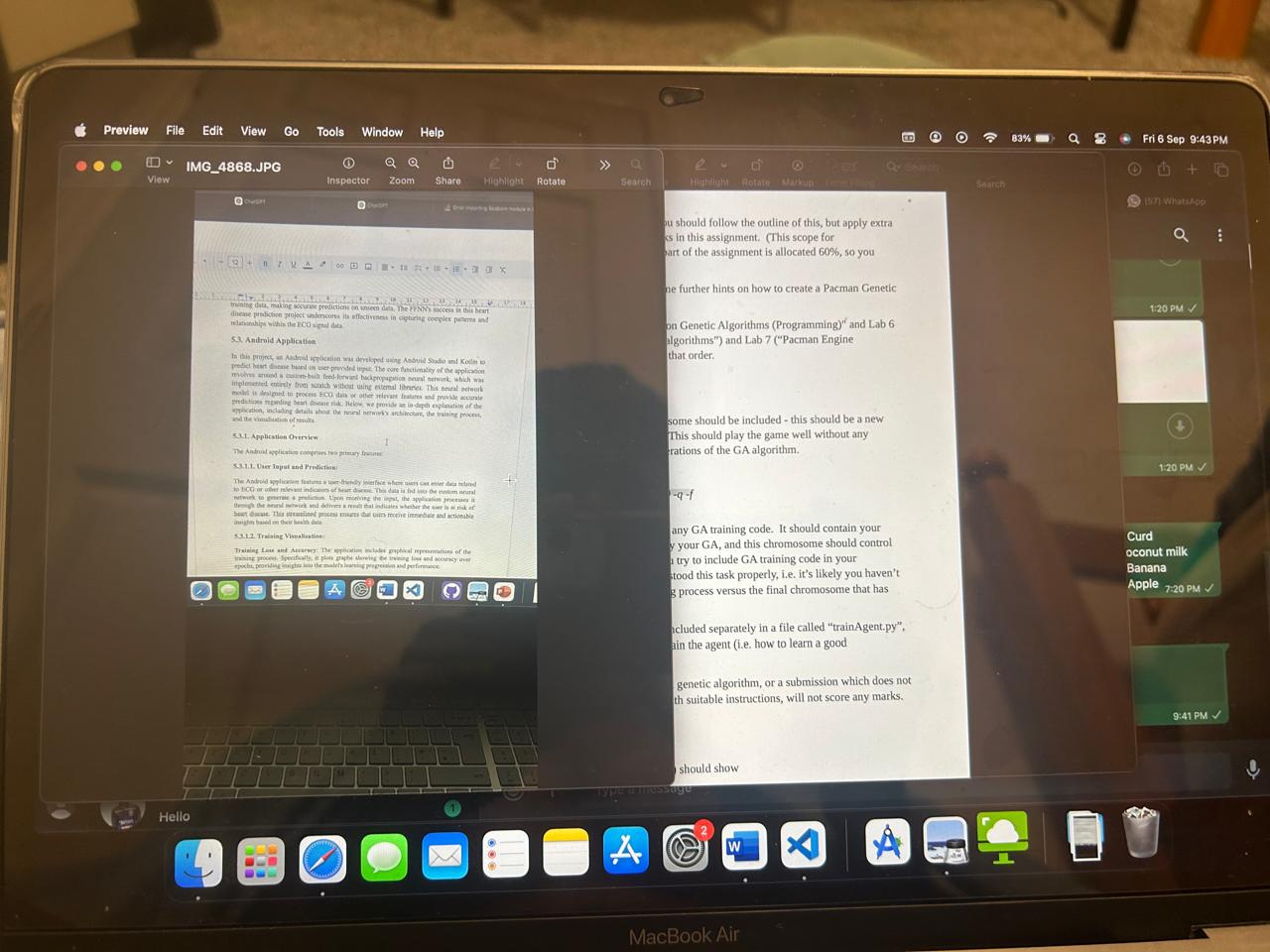
By integrating LIDAR data and a PID controller, the robot can intelligently follow the right edge behaviour. This solution provides a robust and dynamic way of achieving autonomous navigation in cluttered environments.

Right edge following behaviour using fuzzy logic

A new autonomous navigation system designed to follow a right-edge obstacle using **fuzzy logic**. This method enhances the robot's ability to avoid obstacles and navigate smoothly through complex environments.

**Right-Edge Following**

Right-edge following is a navigation strategy where the robot maintains a fixed distance from obstacles on its right-hand side. This enables it to move efficiently along paths while avoiding collisions, particularly useful in indoor settings such as warehouses, homes, or office spaces.



[Upload video]

**The Power of Fuzzy Logic**

Our system uses **fuzzy logic** to handle decision-making, making the robot’s behaviour more flexible and adaptable. Unlike traditional logic systems that operate in binary (yes/no or 0/1), fuzzy logic allows for degrees of truth. This enables smoother and more intuitive navigation, even in uncertain or changing environments.

**How it Works**

The system is powered by a **LiDAR sensor** that continuously scans the surroundings, feeding distance data from different angles to the robot. Here’s how the process works:

* **LiDAR Data Processing**: The robot divides the sensor data into regions (e.g., front, right-front, right-back, left, etc.), providing a comprehensive view of its surroundings.
* **Fuzzy Logic Decision Making**:
  + The distances to obstacles are classified into categories: **near**, **medium**, and **far**.
  + Based on these distance readings, a set of **fuzzy rules** determines how the robot should adjust its movement. For instance, if something is detected close by on the right, the robot will steer left to avoid it. If the right side is clear, it will adjust its path to follow the edge.
  + Through **defuzzification**, the fuzzy system calculates the robot’s **speed** and **angular velocity**, creating smooth transitions between different movement commands. For defuzzification I used Height defuzzification method.
* **Safe and Smooth Navigation**: With this approach, the robot can smoothly adapt its trajectory and speed, avoiding sudden turns or stops while maintaining a safe distance from obstacles.

**Key Features**

1. **Dynamic Obstacle Detection**: Continuously adjusting to the robot's surroundings, making it well-suited for environments where obstacles may move or change.
2. **Efficient Edge Following**: The robot follows the right edge with precision, ensuring it stays on track while avoiding collisions.
3. **Real-Time Adaptation**: The fuzzy logic system updates in real time, ensuring responsive and effective control over the robot’s navigation.
4. **Flexible and Modular**: The solution is adaptable and can be easily integrated with different robot platforms and environments.

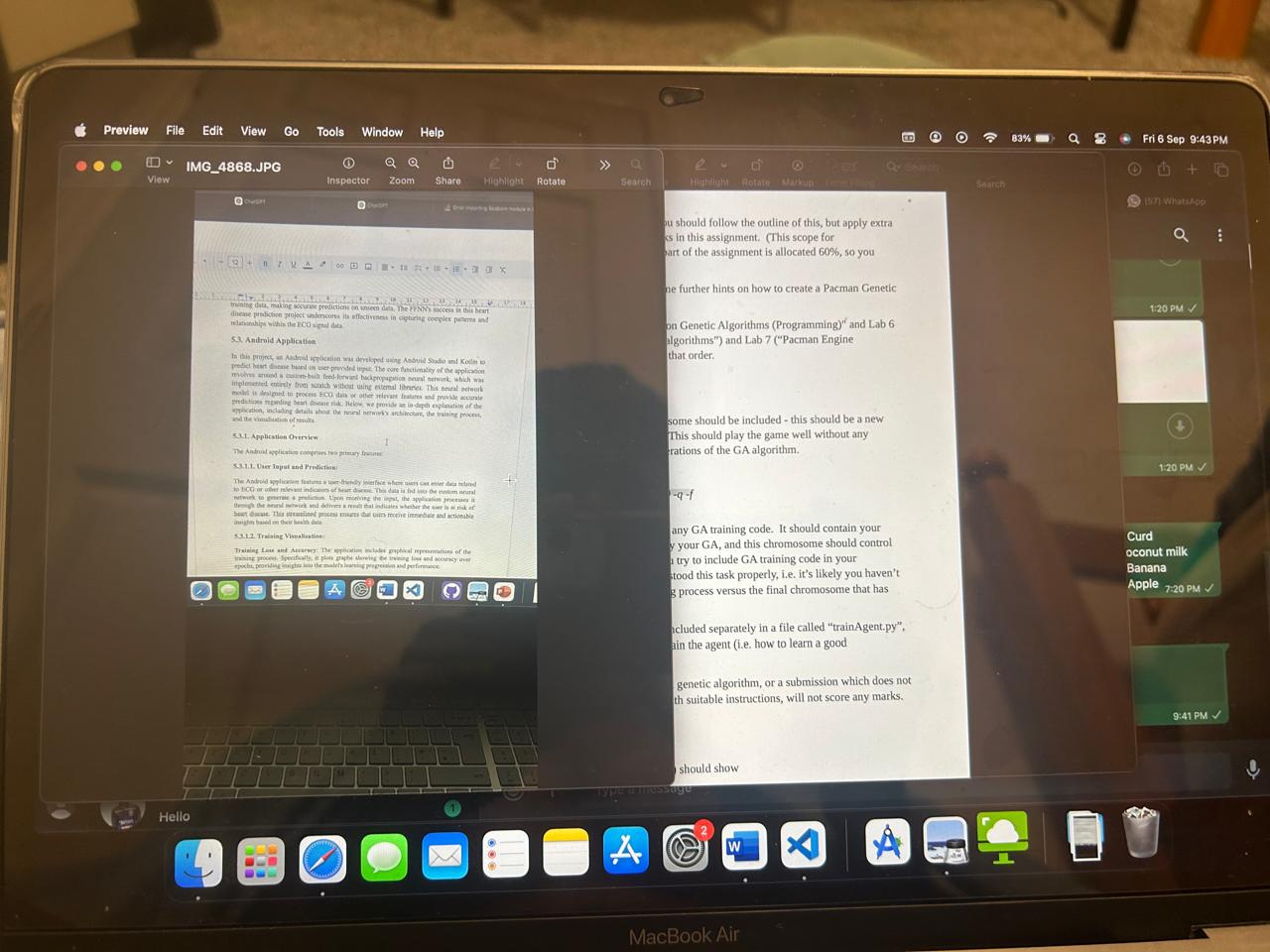
**Code Overview**

This fuzzy logic implementation is built to follow right edge following behaviour, enabling easy integration with robotic platforms and providing robust communication between different components of the system.

Obstacle avoidance behaviour using fuzzy logic

**Overview**

Our state-of-the-art obstacle avoidance system leverages fuzzy logic to navigate complex environments with precision. By integrating three key input sensors and utilizing height defuzzification, our system ensures optimal performance and safety for your robotic applications.



[give the video]

**Key Features**

**1. Fuzzy Logic-Based Obstacle Avoidance**

Our robotic system employs fuzzy logic to make intelligent decisions in real-time, allowing for smooth and effective obstacle avoidance. The fuzzy logic controller processes input from three distinct sensors to assess the proximity of obstacles. This approach helps the robot adapt its navigation strategy dynamically based on environmental conditions.

**2. Three Input Sensors**

The system utilizes three sensors to gather crucial data about the surroundings:

* **Front Sensor:** Measures the distance to obstacles directly in front of the robot.
* **Left Sensor:** Detects obstacles to the left of the robot.
* **Right Sensor:** Monitors obstacles to the right of the robot.

This multi-sensor setup provides comprehensive environmental awareness, allowing the robot to make well-informed navigation decisions.

**3. Height Defuzzification Method**

To convert fuzzy logic outputs into actionable control signals, our system employs the height defuzzification method. This technique helps translate the fuzzy set of rules into precise control commands by considering the height of the membership functions. By focusing on the most relevant values, the robot can make accurate and efficient adjustments to its movement.

**4. Real-Time Adaptability**

The integration of fuzzy logic with height defuzzification ensures that the robot can respond to changes in the environment quickly and effectively. As the robot encounters obstacles, the fuzzy logic controller evaluates the sensor inputs and adjusts the navigation parameters to maintain smooth and safe movement.

**How It Works**

1. **Sensor Data Collection:** The robot's three sensors continuously gather distance information from the environment.
2. **Fuzzy Logic Processing:** The sensor data is processed using fuzzy logic rules, which categorize obstacle proximity and determine the appropriate response.
3. **Height Defuzzification:** The fuzzy output is converted into precise control commands using height defuzzification, ensuring accurate navigation adjustments.
4. **Adaptive Navigation:** The robot adjusts its path and speed based on real-time data, avoiding obstacles and navigating efficiently through complex environments.

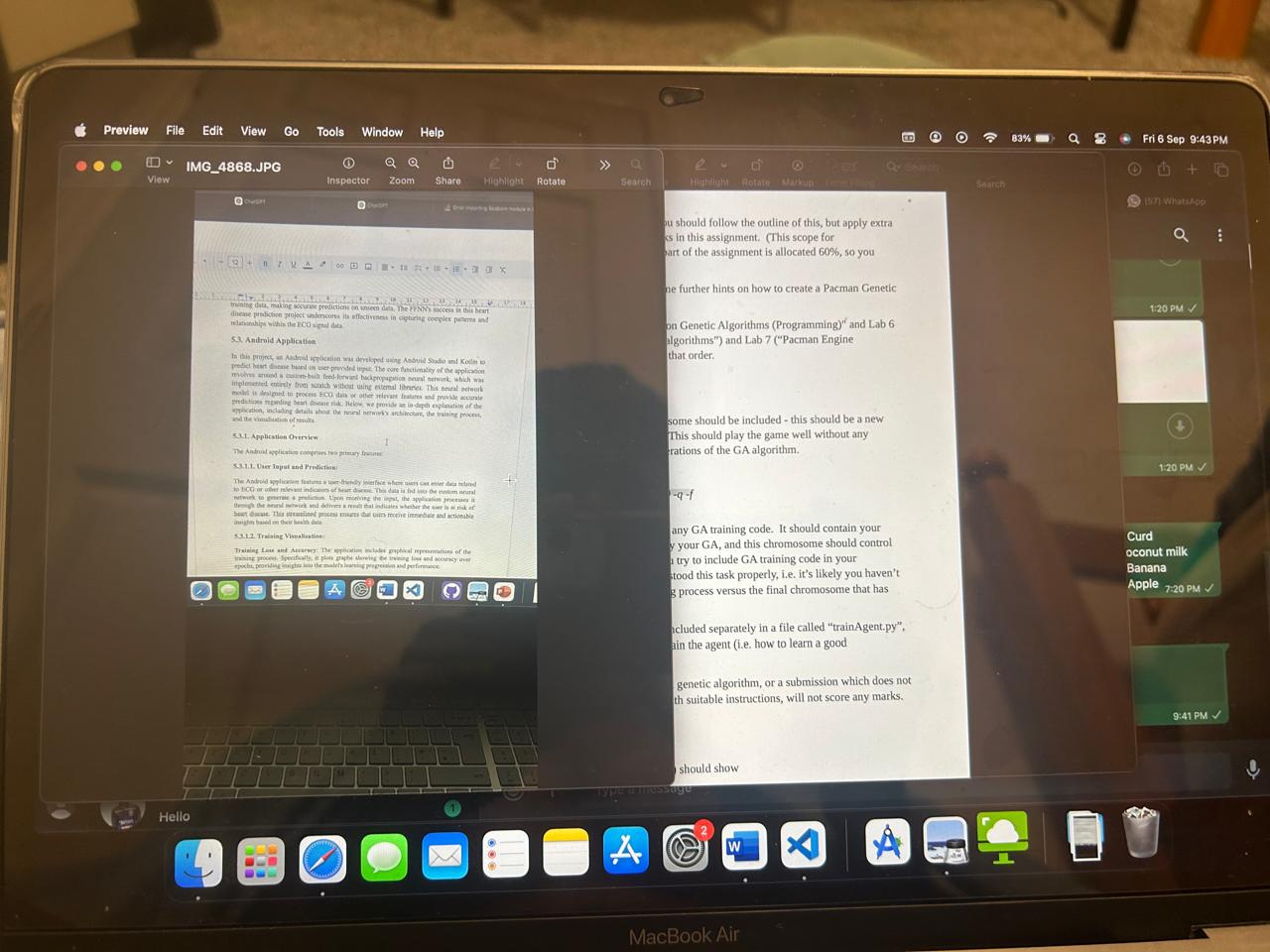
**Applications**

Our fuzzy logic-based obstacle avoidance system is ideal for various applications, including autonomous vehicles, robotic exploration, and industrial automation. Its adaptability and precision make it a powerful tool for ensuring safe and efficient robotic navigation.

Combined obstacle avoidance and right edge using subsumption architecture

**Overview**

This project demonstrates a sophisticated robot control system that combines obstacle avoidance and edge-following behaviours using a subsumption architecture. By prioritizing obstacle avoidance, this system ensures safe navigation in complex environments while still adhering to edge-following rules when obstacles are not detected.



[upload video]

**Key Features**

1. **Subsumption Architecture**
   * **Hierarchy of Behaviours:** The system uses a hierarchical approach to prioritize behaviours. Obstacle avoidance takes precedence over edge-following, and if neither is active, the robot continues straight.
   * **Behaviour Integration:** Combines two distinct behaviours—obstacle avoidance and right edge-following—allowing the robot to adapt dynamically to its surroundings.
2. **Obstacle Avoidance**
   * **Sensor Inputs:** The robot uses LaserScan data from various sensors to detect obstacles.
   * **Decision Making:** When obstacles are detected, the system calculates defuzzified values for direction and speed to navigate around them.
   * **Behaviour Activation:** If the Left Front Sensor (LFS) or Middle Front Sensor (MFS) detects an obstacle, the robot prioritizes avoidance behaviour.
3. **Right Edge Following**
   * **Edge Detection:** When obstacles are not present, the robot uses the Right Front Sensor (RFS) and Right Back Sensor (RBS) to follow the right edge.
   * **Behaviour Activation:** If no obstacles are detected and the right edge is present, the robot switches to edge-following behaviour to navigate along the edge.
4. **Straight Movement**
   * **Default Behaviour:** In the absence of both obstacles and a right edge, the robot moves straight with a predefined speed.

### Achievements

* **Dynamic Navigation:** Implemented a robust control system that dynamically adapts to obstacles and edges.
* **Prioritized Behaviours:** Successfully used subsumption architecture to prioritize safety (obstacle avoidance) over edge-following.
* **Modular Design:** Designed modular functions for obstacle avoidance and edge-following, allowing for flexible adjustments.

Energy availability prediction

### Overview

In this project, I developed a sophisticated climate prediction model using the Colchester weather dataset, which spans from January 2022 to December 2022. The goal was to forecast weather conditions for the next 24 hours with high accuracy by employing a range of machine learning and neural network techniques.

### Data Pre-processing

Effective data pre-processing was crucial for building an accurate predictive model. Here's a summary of the pre-processing steps:

* **Handling Null Values:** I addressed missing data in the dataset by employing the interpolation polynomial method. This technique helped in estimating and filling missing values based on the trend of existing data points.
* **Label Encoding:** Categorical features were converted into numerical values using label encoding, facilitating their use in machine learning models.
* **Data Splitting:** The dataset was divided into training, testing, and validation sets. This step ensured that the models were trained effectively and evaluated on unseen data to gauge their performance.
* **Visualizations:** I performed various visualizations to understand the dataset better, uncovering patterns and trends that informed the modelling process.

### Machine Learning Models

To predict weather conditions, I utilized a diverse set of models, each bringing unique strengths to the prediction task:

* **Linear Regression:** Implemented to understand the relationship between weather variables and forecast outputs.
* **Random Forest:** Used for its ability to handle complex datasets and provide robust predictions by averaging multiple decision trees.
* **Gradient Boosting:** Applied to improve model performance through iterative learning, refining predictions by focusing on errors from previous iterations.
* **XGBoost:** Leveraged for its high performance and efficiency in large-scale datasets, enhancing prediction accuracy with gradient boosting.

### Neural Network Models

In addition to traditional machine learning models, I explored advanced neural network techniques to capture temporal and spatial dependencies in weather data:

* **Long Short-Term Memory (LSTM):** Utilized for its effectiveness in capturing long-term dependencies and patterns in time-series data.
* **Convolutional Neural Networks (CNN):** Employed to detect spatial hierarchies and features within the dataset, enhancing the model's ability to learn complex patterns.

### Results

By integrating these diverse models, I successfully predicted the weather conditions for the next 24 hours. The use of both traditional machine learning and advanced neural network approaches enabled comprehensive and accurate forecasting.

### Key Insights

* The combination of various machine learning models and neural networks provided a robust framework for weather prediction.
* Advanced techniques like LSTM and CNN were instrumental in capturing complex patterns and improving prediction accuracy.
* Effective data pre-processing and visualization played a crucial role in model performance and understanding the dataset.

### Applications

This project demonstrates the potential of machine learning and neural networks in climate prediction. The models developed can be applied to various domains, including weather forecasting, climate research, and environmental monitoring.

Toxic word prediction

### Overview

In this project, I developed a system for predicting toxic words using a combination of machine learning models. The objective was to accurately classify words as toxic or non-toxic by leveraging both **Naive Bayes (Generative)** and **Random Forest (Discriminative)** models. This approach highlights the capabilities of different machine learning techniques in handling text classification tasks.

### Project Structure

* **models/**: Contains the saved machine learning models.
  + **model\_dis/**: Includes the trained Naive Bayes model.
  + **modes\_dis/**: Includes the trained Random Forest model.
* **text\_analysis\_final.ipynb**: A comprehensive Jupyter notebook that encompasses the entire workflow, including data pre-processing, model training, and prediction.

### Pre-processing Steps

The pre-processing phase involved several key steps:

* Removal of stop words and punctuation
* Tokenization of each word

These steps were essential for preparing the dataset for effective model training.

### Models Used

In this project, I employed a range of machine learning and neural network models:

1. **Naive Bayes Model (Generative Approach)**:
   * Utilizes a generative approach to determine whether text is toxic or non-toxic.
   * Trained and saved in the models/model\_dis/ directory.
2. **Random Forest Model (Discriminative Approach)**:
   * Applies a discriminative approach for classification tasks using Random Forest.
   * Trained and saved in the models/modes\_dis/ directory.

Additionally, I explored other models such as Support Vector Machine (SVM), Logistic Regression, Long Short-Term Memory (LSTM), and Convolutional Neural Networks (CNN).

### File Descriptions

* **text\_analysis\_final.ipynb**: Main Jupyter notebook for data pre-processing, model training, and predictions.
* **models/model\_dis/**: Directory with the trained Naive Bayes model.
* **models/modes\_dis/**: Directory with the trained Random Forest model.

### Conclusion

This project showcases the application of **Naive Bayes** and **Random Forest** models in classifying toxic words. By running the provided Jupyter notebook, you can replicate the entire process from data pre-processing to model training and prediction.

Heart disease prediction

### Overview

This project focuses on predicting heart disease using ECG data from the MIT-BIH Arrhythmia Database, a high-resolution dataset annotated by cardiologists. The goal was to classify various cardiac conditions by leveraging a variety of machine learning and neural network models, including innovative techniques for enhanced prediction accuracy.

### Dataset

The MIT-BIH Arrhythmia Database comprises two half-hour ECG recordings from 47 subjects, captured using a two-channel ambulatory instrument. The dataset includes detailed annotations of various cardiac conditions, making it an ideal benchmark for heart disease prognosis models.

### Models and Techniques

#### Traditional Machine Learning Models

* **Random Forest**: Utilized for its ensemble learning capabilities and robust performance.
* **Naive Bayes**: Applied to leverage probabilistic classification.
* **Logistic Regression**: Used for its simplicity and interpretability.
* **Support Vector Machine (SVM)**: Employed to handle complex decision boundaries.
* **Gradient Boosting Classifier**: Applied for boosting model performance through ensemble techniques.

#### Neural Network Models

* **Feedforward Neural Networks**: Used for their ability to model complex patterns in ECG data.
* **Long Short-Term Memory (LSTM) Networks**: Implemented to capture the temporal dynamics and sequence information inherent in ECG signals.

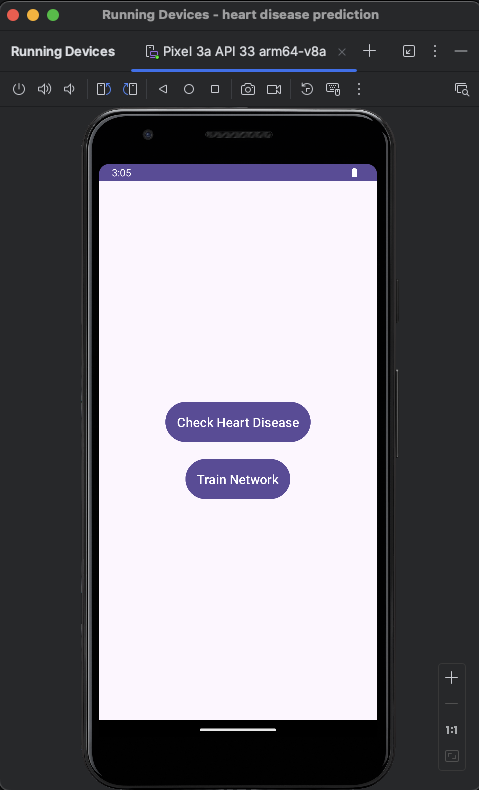
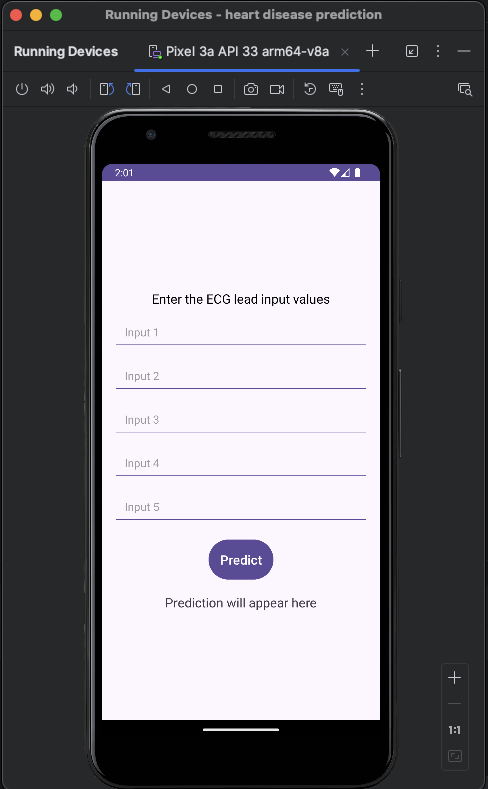
#### Innovative Approaches

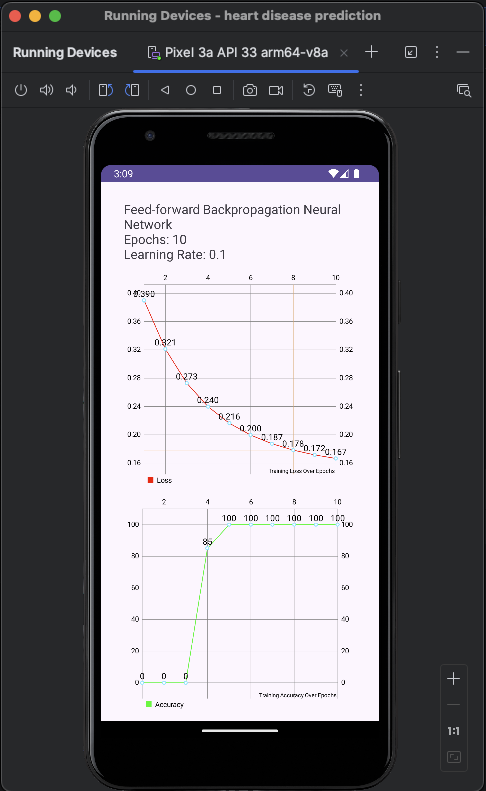
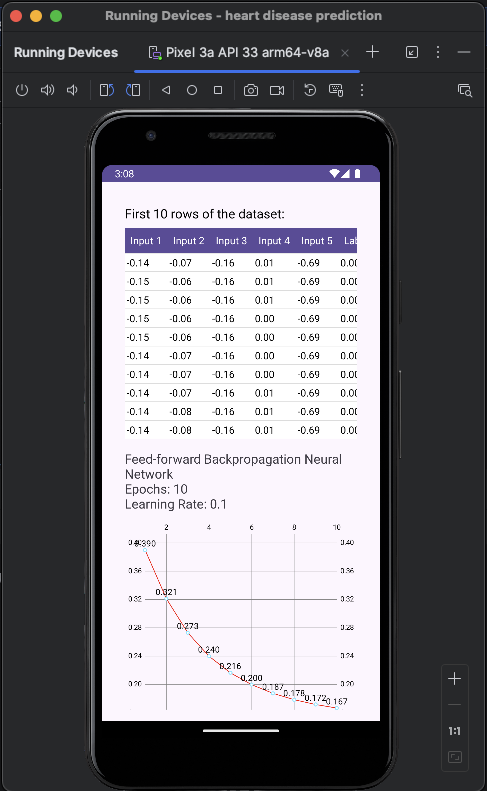
* **Dynamic Neural Network Architecture**: Developed a new adaptive neural network that adjusts its structure based on the complexity of the input data. This architecture aims to provide better predictions compared to static models.
* **Hybrid Model Ensemble**: Combined Logistic Regression with Random Forest and Gradient Boosting Classifier to enhance predictive performance and leverage the strengths of multiple models.

### Android Application

An integral part of the project was the development of an Android application for real-time heart disease prediction. Key features include:

* **Feedforward Neural Network Integration**: Implemented from scratch in Android Studio using Kotlin, without relying on external libraries. This approach allows for seamless integration of the predictive model into a mobile platform.
* **Training and Visualization**: The app includes functionalities to tune neural network parameters, compute accuracy, and monitor training loss. Users can view training progress graphs, which provide insights into model performance and explain the machine learning process.

### Training Procedure

The training of neural network models was conducted directly within the Android application. Key steps included:

* Parameter tuning to optimize model performance
* Accuracy and loss computation to evaluate model effectiveness
* Visualization of training progress to ensure transparency and understanding of the model's behaviour

### Conclusion

This project exemplifies a comprehensive approach to heart disease prediction by integrating traditional machine learning models with advanced neural network techniques. The dynamic neural network architecture and mobile application development demonstrate innovative solutions in real-time heart disease monitoring and prediction. By showcasing these advancements, the project highlights both the potential and practicality of machine learning in healthcare.