SMART SHOPPING CART

A PROJECT REPORT

Submitted by

BHAIRAV S [RA2211003020328]

HARINI V [RA2211003020331]

JEYADEEPAK UR [RA2211003020341]

EVANKA VINOLIYA E [RA2211003020342]

Under the guidance of

Dr. V GOWRI

(Assistant Professor, Department of Computer Science and Engineering)

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM, CHENNAI -600089

OCTOBER 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University U/S 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled **"SMART SHOPPING CART"** is the bonafide work of **BHAIRAV S [RA2211003020328], HARINI V [RA2211003020331], JEYADEEPAK UR [RA2211003020341], EVANKA VINOLIYA E [RA2211003020342]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an occasion on this or any other candidate.

SIGNATURE	SIGNATURE
Dr. V GOWRI	**Dr. K. RAJA, M.E., Ph.D.,**
Assistant Professor	**Professor and Head**
Computer Science and Engineering,	Computer Science and Engineering,
SRM Institute of Science and Technology,	SRM Institute of Science and Technology,
Ramapuram, Chennai.	Ramapuram, Chennai.

Submitted for the project viva-voce held on ___________ at SRM Institute of Science and Technology, Ramapuram, Chennai -600089.

INTERNAL EXAMINER 1 **INTERNAL EXAMINER 2**

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
RAMAPURAM, CHENNAI - 89

DECLARATION

We hereby declare that the entire work contained in this project report titled "**SMART SHOPPING CART**" has been carried out by **BHAIRAV S [RA2211003020328], HARINI V [RA2211003020331], JEYADEEPAK UR [RA2211003020341], EVANKA VINOLIYA E [RA2211003020342]** at SRM Institute of Science and Technology, Ramapuram Campus, Chennai- 600089, under the guidance of **Dr. GOWRI Assistant Professor**, Department of Computer Science and Engineering.

Place: Chennai

Date:

BHAIRAV S

HARINI V

JEYADEEPAK UR

EVANKA VINOLIYA E

ACKNOWLEDGEMENT

We place on record our deep sense of gratitude to our lionized Chairman **Dr. R. SHIVAKUMAR** for providing us with the requisite infrastructure throughout the course.

We take the opportunity to extend our hearty and sincere thanks to our **Dean, Dr. M. SAKTHI GANESH, Ph.D.**, for maneuvering us into accomplishing the project.

We take the privilege to extend our hearty and sincere gratitude to the Professor and Head of the Department, **Dr. K. RAJA, M.E., Ph.D.,** for his suggestions, support and encouragement towards the completion of the project with perfection.

We express our hearty and sincere thanks to our guide **Dr. GOWRI,** Assistant Professor, Computer Science and Engineering Department for her encouragement, consecutive criticism and constant guidance throughout this project work.

Our thanks to the teaching and non-teaching staff of the Computer Science and Engineering Department of SRM Institute of Science and Technology, Ramapuram, for providing necessary resources for our project.

ABSTRACT

Traditional shopping carts and checkout processes often present significant inefficiencies and challenges for shoppers, making it difficult for them to remember additional items and navigate complex store layouts. This inefficiency affects a broad spectrum of customers in This issue underscores the need for a more effective solution that enhances the efficiency and convenience of the shopping process. To address these concerns, we propose an innovative smart shopping cart system that aims to transform the traditional shopping experience. When shoppers provide their shopping purpose, the cart generates an editable to-do list that assists them in tracking and achieving their shopping goals. If users choose to skip this step, the smart cart utilizes advanced technology to offer contextual product suggestions based on the items currently detected in the cart. the cart ensures that recommendations are accurate and relevant, enhancing the user's shopping experience. Our system begins by prompting users to enter their shopping purpose, though this step is optional. the cart generates an editable to-do list that assists them in tracking and achieving their shopping goals. If users choose to skip this step, the smart cart utilizes advanced technology to offer contextual product suggestions based on the items currently detected in the cart. our smart cart system simplifies the checkout process. Upon parking the cart, it automatically generates a detailed bill, providing users with a comprehensive summary of their purchases. The system also offers flexible payment options to accommodate various preferences and expedite the checkout process. this approach aims to address common pain points in shopping, improve purchase efficiency, and significantly elevate customer satisfaction by delivering a more personalized, streamlined, and efficient shopping experience.

TABLE OF CONTENTS

ABSTRACT		**iv**
S.NO	**TITLE**	**Page No.**
1.	**INTRODUCTION**	**1**
2.	**LITERATURE SURVEY**	**3**
3.	**PROPOSED METHODOLOGY**	**4**
4.	**RESULTS AND DISCUSSION**	**11**
5.	**CONCLUSION**	**17**
6.	**FUTURE ENHANCEMENTS**	**18**
7.	**SOURCE CODE**	**19**
8.	**REFERENCES**	**33**
9.	**PUBLICATION PROOF**	**34**
10.	**PLAGIARISM PROOF**	**35**

Chapter 1

INTRODUCTION

1.1 Introduction

Traditional shopping carts and checkout processes often fall short, leading to inefficiencies such as forgetting additional items and navigating complex store layouts. These challenges can result in incomplete purchases and a frustrating shopping experience. To address these issues, we propose a smart shopping cart system designed to enhance convenience and efficiency. This system prompts users for their shopping purpose and, if provided, generates an editable to-do list. If skipped, it offers contextual product suggestions based on items detected in the cart. Additionally, it streamlines the checkout process by generating a detailed bill and offering flexible payment options. This report explores the development and benefits of this innovative system, aiming to improve shopping efficiency and customer satisfaction.

Traditional shopping carts and checkout processes in supermarkets and grocery stores often present significant challenges and inefficiencies. Shoppers frequently struggle with remembering additional items they need to purchase and navigating complex store layouts, which can lead to incomplete or suboptimal purchases. These inefficiencies are compounded by the lack of personalized assistance, resulting in customers overlooking complementary products and failing to align their purchases with specific shopping goals. As a result, the shopping experience becomes frustrating and inefficient, impacting overall customer satisfaction and purchase efficiency. To address these issues, there is a need for an advanced solution that enhances the shopping process. This solution should facilitate better item management, provide contextual product suggestions, and streamline the checkout experience. By integrating real-time inventory data and offering personalized assistance, the solution aims to improve both the efficiency of the shopping process and customer satisfaction.

The objective of the Smart-Shopping-Cart Web App is to enhance the shopping experience by providing users with a personalized and efficient shopping process. The application aims to

1.Purpose Identification: Allow users to specify the purpose of their visit through an intuitive interface, with an option to skip this step if desired.

2.Checklist Generation: Automatically generate a relevant checklist based on the entered purpose, enabling users to review, edit, and proceed with their shopping more effectively.

3.Product Recommendations: Offer product suggestions based on items scanned into the cart, ensuring users receive relevant recommendations and can make informed purchase decisions.

4.Seamless Transition: Provide a smooth transition from the purpose entry to checklist management or product recommendations, ensuring a user-friendly and streamlined shopping experience.

This project is situated within the domain of smart retail and machine learning. It involves the application of data science techniques to enhance the shopping experience, with a particular focus on predictive modeling to offer personalized product recommendations. The project incorporates data preprocessing, user behavior analysis, and the implementation of machine learning algorithms to generate relevant checklists and product suggestions. By leveraging these techniques, the smart-shopping-cart app aims to provide a more efficient and tailored shopping experience for users. The scope of the Smart-Shopping-Cart project includes developing a user-friendly application that enhances the shopping experience through personalized checklists and product recommendations. The project will feature a purpose entry system that allows users to specify their shopping goals or skip this step, followed by the generation of a relevant checklist. If the purpose is skipped, the app will suggest products based on scanned items. The scope encompasses designing the user interface using Tkinter, implementing data preprocessing, and applying machine learning algorithms for accurate recommendations. Testing, documentation, and user feedback collection are also part of the project to ensure functionality and effectiveness. Advanced features such as real-time tracking or system integrations are not included.

Chapter 2

LITERATURE REVIEW

The Literature Review examines the convergence of smart retail technologies and machine learning, emphasizing their role in revolutionizing the shopping experience. Smart retail technologies have evolved to include various innovations such as automated checkout systems, smart shopping carts, and in-store analytics. These technologies aim to enhance operational efficiency and improve customer satisfaction by integrating real-time data and advanced automation. For instance, automated checkout systems, such as those pioneered by Amazon Go, use computer vision and sensor fusion to enable cashier-less transactions, reducing wait times and streamlining the checkout process. Machine learning plays a crucial role in advancing these technologies, particularly through predictive modeling and recommendation systems. Machine learning algorithms analyze historical and real-time data to provide personalized recommendations, optimizing product suggestions based on user preferences and shopping behavior. Techniques such as collaborative filtering, content-based filtering, and hybrid models are employed to enhance the accuracy of these recommendations. Research has shown that personalized recommendations can significantly increase customer satisfaction and sales by aligning product suggestions with individual user interests.

Furthermore, user experience (UX) design is integral to the success of retail applications. Effective UX design ensures that interfaces are intuitive, engaging, and responsive to user needs. In the context of smart retail, UX design focuses on creating seamless interactions between users and technology. Research highlights that a well-designed user interface can improve user satisfaction and adoption rates, making it a critical component in the development of retail applications. This Literature Review provides a comprehensive overview of how smart retail technologies and machine learning intersect to enhance the shopping experience. It sets the stage for understanding the technological advancements and design principles that inform the development of the Smart-Shopping-Cart project, offering insights into the current state of the field and identifying areas for further innovation.

Chapter 3

PROJECT METHODOLOGY

The existing systems in the realm of smart retail predominantly include traditional shopping carts with limited integration of technology, basic self-checkout kiosks, and rudimentary mobile shopping apps. These systems often lack advanced personalization and real-time assistance capabilities. For instance, traditional shopping carts require manual list management, offering no real-time suggestions or intelligent features. Self-checkout systems, while reducing wait times, primarily focus on the payment process without enhancing the overall shopping journey. Mobile shopping apps may provide some level of product recommendations, but they are generally based on simple algorithms with limited contextual awareness. Additionally, these systems often operate in isolation, without a cohesive integration that ties the shopping experience together from list creation to purchase. Consequently, the current systems fall short in offering a fully personalized, interactive, and efficient shopping experience, highlighting the need for more advanced solutions like the Smart-Shopping-Cart that leverages machine learning and smart technology to fill these gaps.

The proposed Smart-Shopping-Cart system aims to revolutionize the shopping experience by integrating advanced machine learning algorithms and smart technology into the shopping process. Unlike existing systems, this application offers a personalized and interactive approach, starting with an optional purpose entry feature that allows users to specify their shopping goals. Based on the provided purpose, the system generates a customized checklist, which users can edit before proceeding with their shopping. If the purpose is skipped, the application uses machine learning to analyze scanned items and provide relevant product recommendations in real-time. The system's recommendation engine leverages predictive modeling to suggest complementary products, ensuring a more tailored and efficient shopping experience. Additionally, the user interface is designed with ease of use in mind, utilizing Tkinter to create a seamless and intuitive navigation flow. By combining these features, the proposed system significantly enhances user satisfaction and streamlines the shopping process, addressing the limitations of current solutions.

The feasibility of the Smart-Shopping-Cart system is supported by several factors, including technical, economic, and operational considerations. **Technically**, the project leverages widely used technologies such as Python’s Tkinter for the user interface and established machine learning algorithms for product recommendations, making the development process achievable with current tools and knowledge. **Economically**, the system offers a cost-effective solution by enhancing shopping efficiency and customer satisfaction, potentially leading to higher sales and customer retention for retailers. The initial development costs are justified by the long-term benefits of improved user experience and operational efficiency. **Operationally**, the system can be easily integrated into existing retail environments with minimal disruption, as it enhances rather than replaces current shopping processes. The user-friendly interface ensures that both customers and store staff can quickly adapt to the new system. Overall, the Smart-Shopping-Cart system is a viable project with strong potential for successful implementation and positive impact on the retail experience.

The economic feasibility of the Smart-Shopping-Cart system is promising, given its potential to deliver significant returns on investment through enhanced customer experiences and increased sales. The initial development costs, including software development, machine learning model training, and user interface design, are manageable and can be minimized by using open-source tools like Python and Tkinter. Once implemented, the system's ability to provide personalized recommendations can lead to higher purchase volumes and improved customer retention, directly translating into increased revenue for retailers. Additionally, by streamlining the shopping process and reducing the likelihood of forgotten items, the system can encourage repeat business and foster customer loyalty. The reduced need for manual assistance in-store also means potential labor cost savings. Over time, the Smart-Shopping-Cart system is expected to generate more revenue than the costs incurred in its development and maintenance, making it an economically viable solution for modern retail environments.

The technical feasibility of the Smart-Shopping-Cart system is highly favorable due to the availability of mature technologies and development tools. The system is built using Python, a versatile and widely supported programming language, with Tkinter for creating the user interface, ensuring compatibility across different platforms. The use of established machine learning algorithms for generating product recommendations is well within the capabilities of current computational resources. Additionally, the required hardware components, such as barcode scanners and basic sensors for item detection, are readily available and can be easily integrated into the system. Data processing and storage requirements can be managed with existing cloud services or local databases, making the infrastructure scalable as needed. Given the robustness of the tools and technologies involved, the project is technically feasible and can be developed and deployed with a high degree of confidence in its performance and reliability

Advantages

The proposed Smart-Shopping-Cart system offers several key advantages over existing solutions, significantly enhancing the shopping experience. One of the primary benefits is its ability to provide personalized and context-aware recommendations, leveraging machine learning algorithms to suggest relevant products based on user input or scanned items. This not only improves shopping efficiency but also helps users discover items they might otherwise overlook. The system's purpose entry and checklist generation features allow for a more organized and goal-oriented shopping trip, reducing the chances of missing essential items. Additionally, the user-friendly interface, built with Tkinter, ensures that the system is accessible and easy to navigate, catering to users of all tech-savviness levels. By integrating these smart technologies, the proposed system not only streamlines the shopping process but also enhances user satisfaction, making it a powerful tool in the evolving landscape of smart retail.

General Architecture:

The general architecture of the Smart-shopping-Cart system is designed to integrate seamlessly with the retail environment, consisting of multiple interconnected layers to ensure smooth operation and user interaction. At the core is the Application Layer, which includes the primary software components such as the user interface built with Tkinter, the machine learning models for product recommendation, and the database management system (SQLite). This layer directly interacts with the Hardware Layer, encompassing the barcode scanner, touchscreen display, and processing unit, which handle the physical input and output operations. The Data Layer manages the storage and retrieval of user data, product information, and shopping history, ensuring that all necessary information is readily accessible for processing and recommendations. Optionally, the architecture may include a Network Layer for cloud integration, allowing for remote data storage, updates, and advanced analytics. The Security Layer is integrated across all other layers to protect user data and ensure compliance with privacy regulations. This layered architecture ensures that the system is modular, scalable, and capable of delivering a responsive and personalized shopping experience while maintaining security and data integrity.

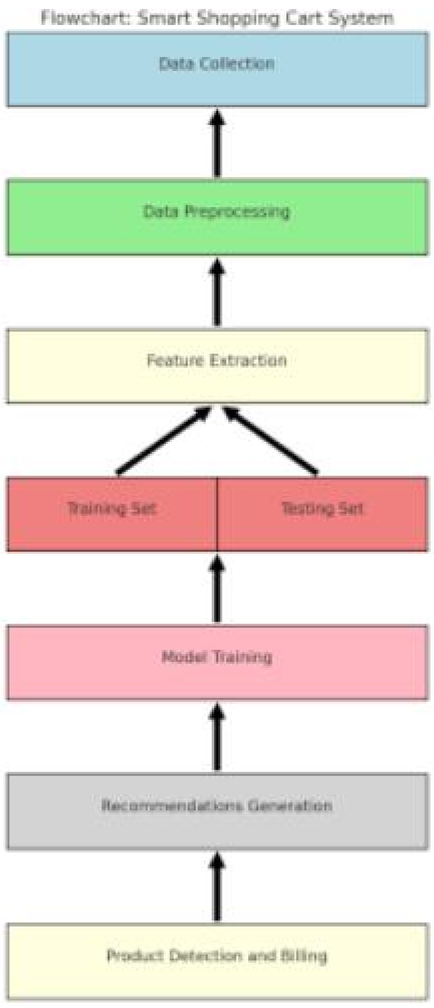

Flowchart: Smart Shopping Cart System
Data Collection
Data Preprocessing
Feature Extraction
Training Set
Testing Set
Model Training
Recommendations Generation
Product Detection and Billing

Figure 3.1: **Architecture Diagram**

Figure 3.1 This flowchart represents the architecture of a Smart Shopping Cart System. The process starts with Data Collection, followed by Data Preprocessing to clean and organize the data. After that, Feature Extraction is performed to identify key information, which is then split into Training and Testing Sets. The system undergoes Model Training

to learn patterns from the data, enabling Recommendations Generation. Finally, the system integrates Product Detection and Billing to provide seamless shopping experiences.

Figure 3.2 This data flow diagram illustrates the process of a smart shopping system using RFID technology. The system starts by initializing and scanning products with an RFID reader. If a product is scanned, the item details are displayed on an LCD. The product amount is added to the system, and if the item is removed, it is also updated in real-time on a web application. The system continuously checks for added or removed items, and once the shopping is complete, it prints the bill and stops the process.

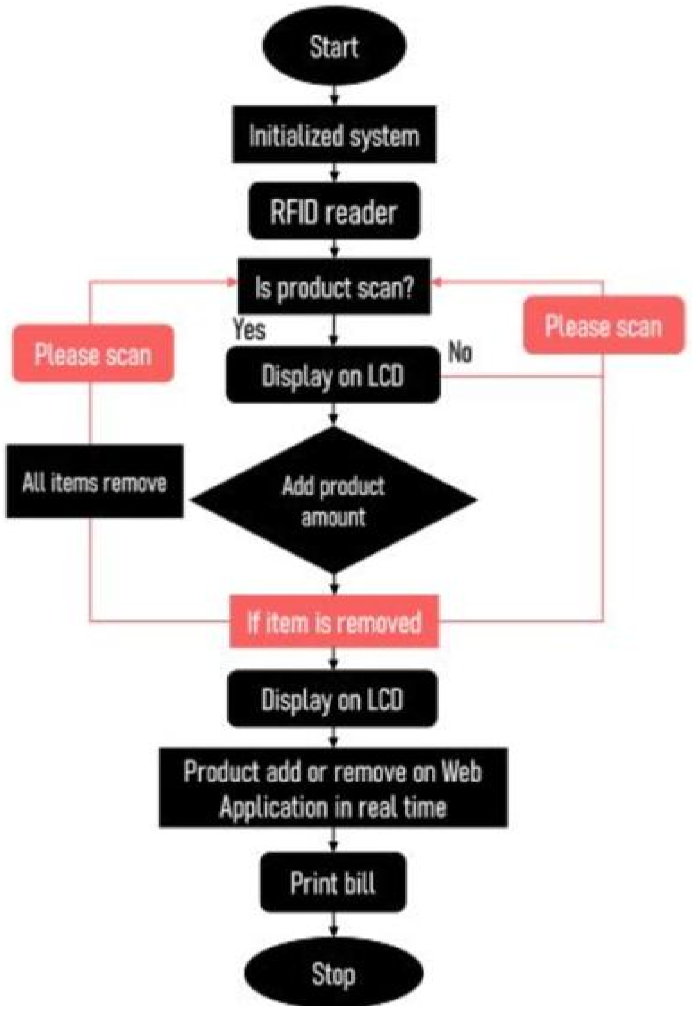


Figure 3.2: **Data Flow Diagram**

Preprocessing

In developing the smart-shopping-cart web app, data collection plays a crucial role in tailoring the shopping experience to each user. Initially, product images are gathered from retailer databases, in-store photography, and public datasets to build a comprehensive visual library. User interaction data is then collected through in-store sensors and simulated environments, capturing how customers engage with the cart, whether by entering their shopping purpose or by directly scanning items. Feedback from test users is also collected through forms and surveys to understand preferences and refine interactions. Additionally, tracking systems and manual logging methods are employed to record user actions, ensuring the app's recommendation system can suggest relevant products based on scanned items, creating a seamless and personalized shopping journey.

Object detection

The product detection in the Smart-Shopping-Cart system is achieved through an integrated camera that captures images of the items as they are placed in the cart. Utilizing advanced image recognition technology, the system processes these images in real-time, identifying products based on their visual characteristics such as shape, color, and packaging. The camera feeds the captured images into a machine learning model, trained on a comprehensive dataset of product images, to accurately classify and recognize each item. Once a product is detected, the system cross-references it with a product database to retrieve detailed information, including the product name, category, and price, which are then displayed on the user interface. This camera-based detection method eliminates the need for manual scanning, allowing for a more seamless and intuitive shopping experience. Additionally, this approach enables the system to detect multiple items simultaneously and even recognize products without traditional barcodes, enhancing the efficiency and accuracy of the shopping process.

The detection of specific directional motion in the Smart-Shopping-Cart system is implemented using motion sensors and computer vision techniques. The system is equipped with cameras and motion sensors that monitor the movement of items within the cart, as well as the cart's movement through the store. These sensors detect motion in different directions—forward, backward, left, and right—allowing the system to interpret user actions, such as adding or removing items from the cart. The camera, combined with image processing algorithms, tracks the motion of objects relative to the cart and recognizes specific gestures or actions, such as placing an item in the cart or moving it to a different section.. The ability to detect and interpret directional motion enhances the system's responsiveness and accuracy, contributing to a more interactive and intuitive shopping experience.

Recommendation

The recommendation system in the smart shopping cart plays a pivotal role in enhancing the overall shopping experience by providing personalized and contextually relevant suggestions to customers. This system leverages machine learning and data-driven techniques to analyze the user's shopping behavior, selected items, and potential needs, offering product recommendations that align with their preferences or intended purchases.

The recommendation system in the smart shopping cart is built to seamlessly integrate into the user’s shopping journey. It is activated in two ways: either when the user skips the purpose identification step or when they scan an item into the cart. Upon scanning or detecting items, the system processes this information and begins suggesting related products that complement the already selected items. For example, if a customer adds chicken, rice, and curd to the cart, the system can suggest complementary products such as biryani masala, specific spices, or even side dishes that pair well with their meal plan.

This ability to suggest items not only reduces the risk of customers forgetting essential components of their intended purchases, but it also helps them discover new products they might not have considered otherwise. These suggestions are powered by algorithms that analyze patterns in user behavior, popular combinations of products, and data from other users’ shopping habits.

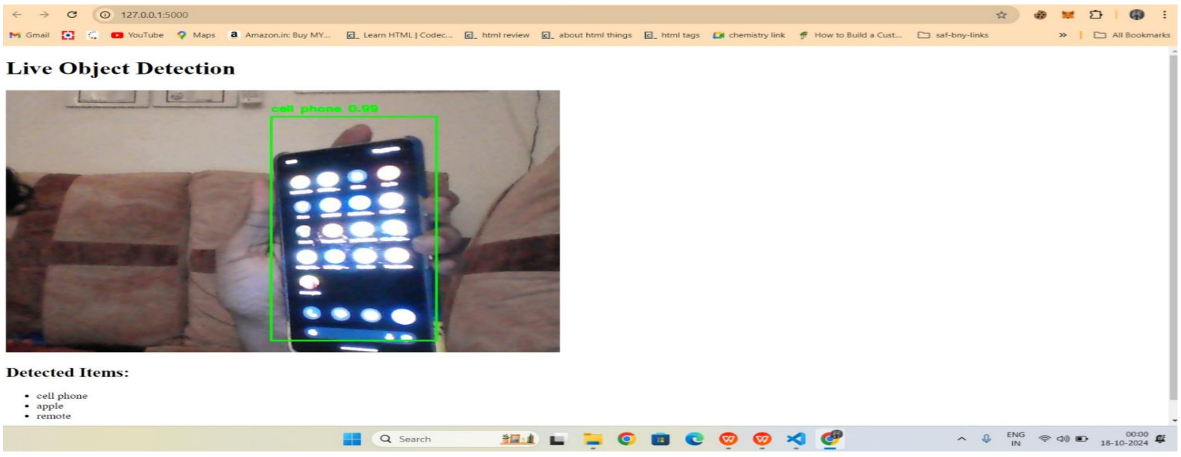
The Role of Machine Learning

The recommendation system relies heavily on machine learning techniques to ensure that the suggestions it offers are relevant and beneficial to the customer. Through continuous learning and improvement, the system can adapt to each user’s unique preferences over time. Initially, the recommendations might be based on general trends, but as the system gathers more data on individual shopping patterns, it can provide increasingly personalized suggestions.
Collaborative filtering, for example, could be used to identify product pairs that are frequently purchased together by other customers. If many users have previously bought biryani masala alongside chicken, the system would detect this pattern and apply it when

Chapter 4

RESULTS AND DISCUSSION

4.1 Input and Output

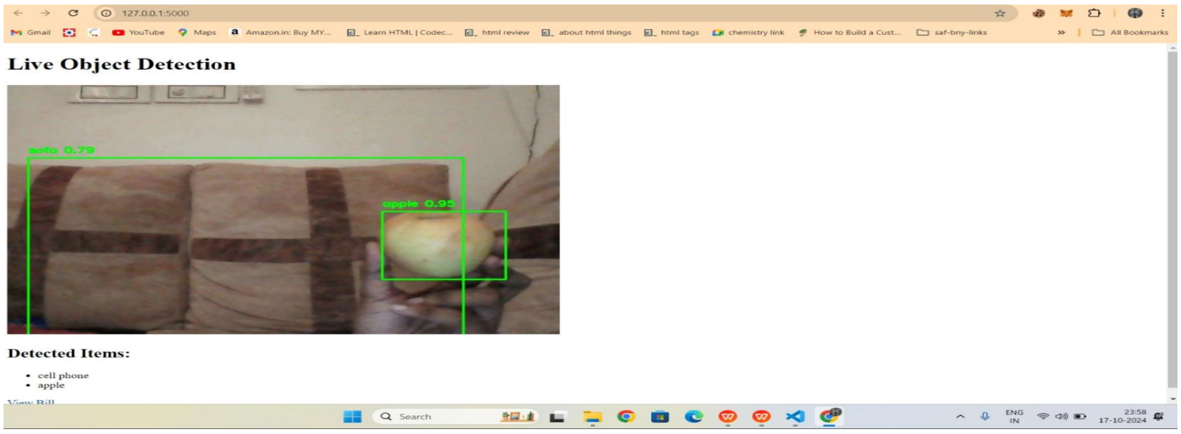


Live Object Detection

Detected Items:

- cell phone
- apple
- remote

Figure 4.1: **Output 1**



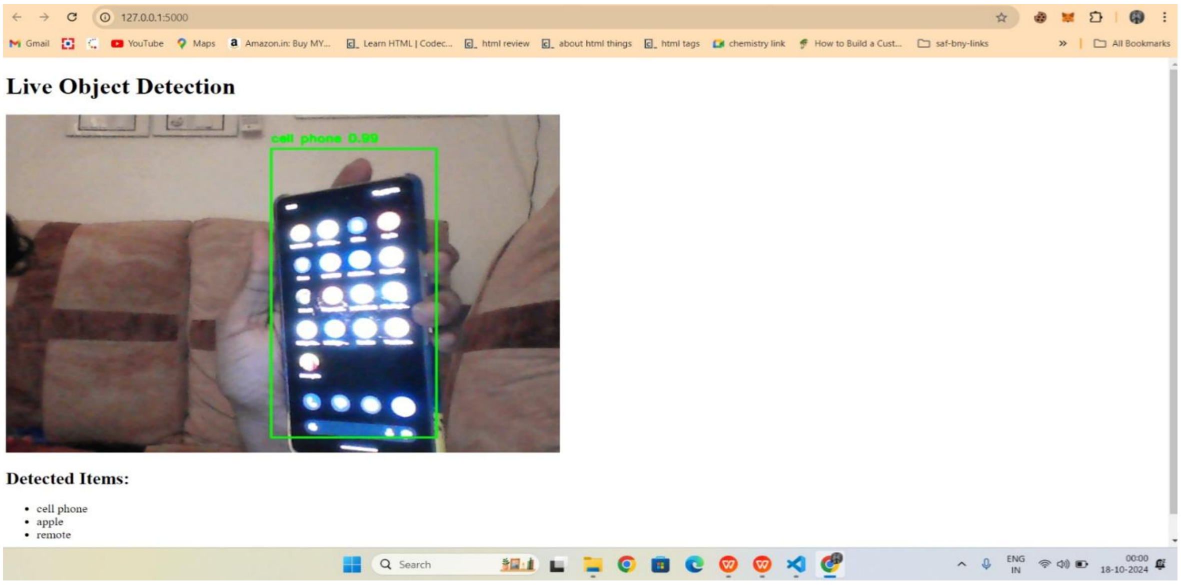
Live Object Detection

Detected Items:

- cell phone
- apple

View Bill

Figure 4.2: **Output**

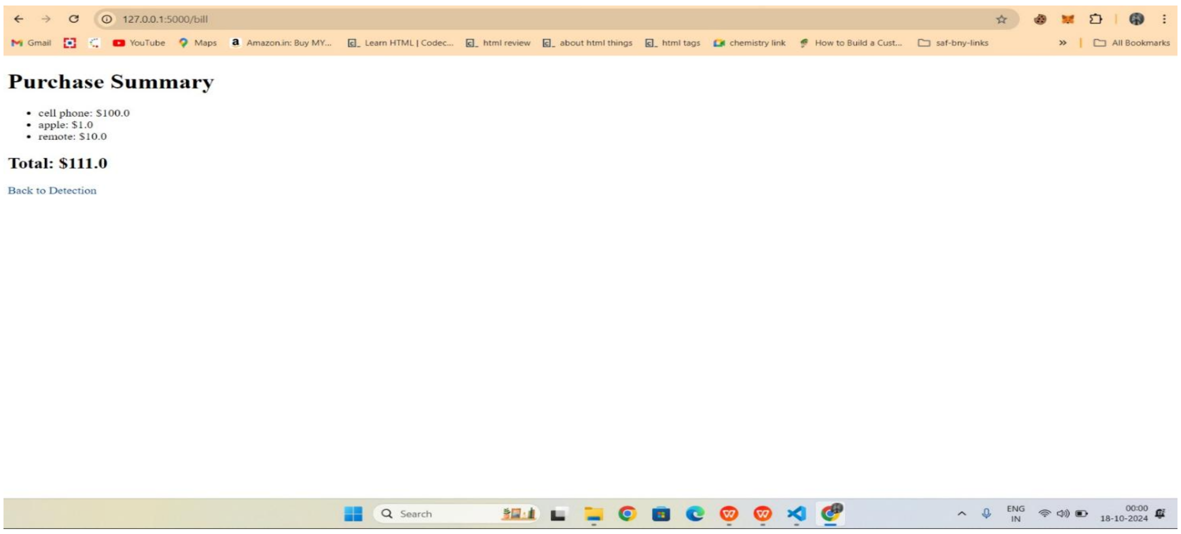


Live Object Detection

Detected Items:

- cell phone
- apple
- remote

Figure 4.3: **Output 3**



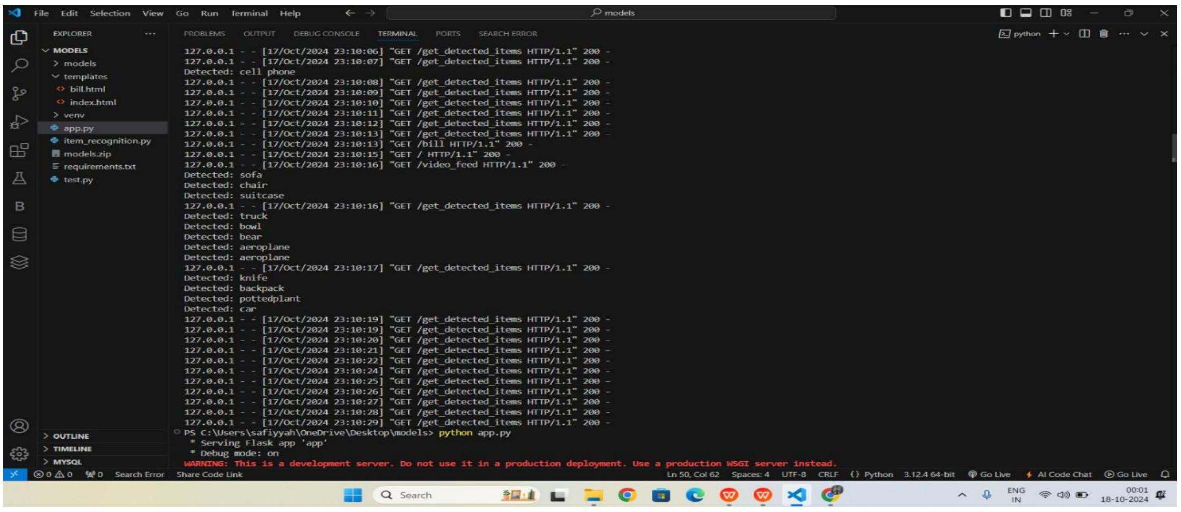
Purchase Summary

- cell phone: $100.0
- apple: $1.0
- remote: $10.0

Total: $111.0

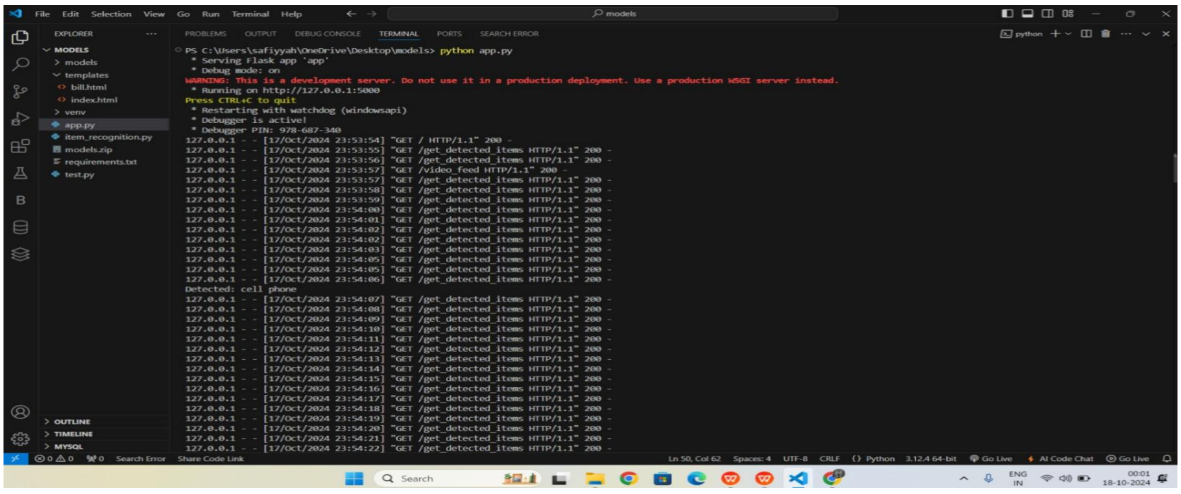
Back to Detection

Figure 4.4: Output 4



```
127.0.0.1 - - [17/Oct/2024 23:10:06] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:07] "GET /get_detected_items HTTP/1.1" 200 -
Detected: cell phone
127.0.0.1 - - [17/Oct/2024 23:10:08] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:09] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:10] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:11] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:12] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:13] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:13] "GET /bill HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:15] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:16] "GET /video_feed HTTP/1.1" 200 -
Detected: sofa
Detected: chair
Detected: suitcase
127.0.0.1 - - [17/Oct/2024 23:10:16] "GET /get_detected_items HTTP/1.1" 200 -
Detected: truck
Detected: bowl
Detected: bear
Detected: aeroplane
Detected: aeroplane
127.0.0.1 - - [17/Oct/2024 23:10:17] "GET /get_detected_items HTTP/1.1" 200 -
Detected: knife
Detected: backpack
Detected: pottedplant
Detected: car
127.0.0.1 - - [17/Oct/2024 23:10:19] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:19] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:20] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:21] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:22] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:24] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:25] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:26] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:27] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:28] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:10:29] "GET /get_detected_items HTTP/1.1" 200 -
PS C:\Users\safiyyah\OneDrive\Desktop\models> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
```

Figure 4.5: Output 5



```
PS C:\Users\safiyyah\OneDrive\Desktop\models> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 978-687-340
127.0.0.1 - - [17/Oct/2024 23:53:54] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:53:55] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:53:56] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:53:57] "GET /video_feed HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:53:57] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:53:58] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:53:59] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:00] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:01] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:02] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:02] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:03] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:05] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:05] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:06] "GET /get_detected_items HTTP/1.1" 200 -
Detected: cell phone
127.0.0.1 - - [17/Oct/2024 23:54:07] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:08] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:09] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:10] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:11] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:12] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:13] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:14] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:15] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:16] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:17] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:18] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:19] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:20] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:21] "GET /get_detected_items HTTP/1.1" 200 -
127.0.0.1 - - [17/Oct/2024 23:54:22] "GET /get_detected_items HTTP/1.1" 200 -
```

Figure 4.6: Output 6

4.2Efficiency of Proposed System

The Smart Shopping Cart system is designed to revolutionize the traditional shopping experience by integrating advanced technologies such as machine learning, deep learning,. It streamlines the shopping process and enhances convenience for customers while improving operational efficiency for retailers. Below, we explore the efficiency of the proposed system across various dimensions.

- **Time Efficiency**: The smart shopping cart significantly reduces the time customers spend on tasks such as finding products, creating shopping lists, and checking out. The system allows users to skip the manual process of searching for items in-store by suggesting products based on previously added items or the purpose of their visit. Additionally, the checkout process is streamlined through automated billing, which minimizes the need to stand in long queues. The smart cart keeps track of products added, generates the bill, and offers easy payment methods
- **Data-Driven Personalization:** A major strength of the smart shopping cart system is its ability to offer personalized experiences. By analyzing a customer's purchasing history and detecting the items added during a shopping session, the system recommends products that align with their preferences or dietary needs. This personalization improves the likelihood of customers finding what they want quickly, thereby boosting their satisfaction and encouraging loyalty to the store. Furthermore, the ability to input the purpose of a visit (e.g., hosting a dinner, weekly groceries) or to skip the step and receive automated suggestions based on cart items creates a balance between structured and dynamic shopping experiences. These features allow the system to cater to different shopping styles and needs efficiently.
- **Improved Customer Experience:** The seamless integration of machine learning and IoT in the smart shopping cart offers a user-friendly interface that elevates the shopping experience. The ability to create or edit shopping lists based on the purpose of the visit enhances the experience for customers, allowing them to plan their purchases more effectively. By minimizing manual tasks and reducing checkout delays, the system fosters a smooth, hassle-free shopping journey, ultimately leading to higher customer satisfaction.

- Scalability and Flexibility: The smart shopping cart system is scalable, meaning it can be easily adapted to stores of various sizes—from small supermarkets to large hypermarkets. The system’s algorithms can be tailored to specific retail environments, optimizing performance based on customer traffic, inventory, and product types.The flexibility of the system also allows it to be integrated with other services such as loyalty programs, personalized promotions, and digital coupons. This not only makes shopping more efficient but also opens new avenues for retailers to engage with customers and drive revenue

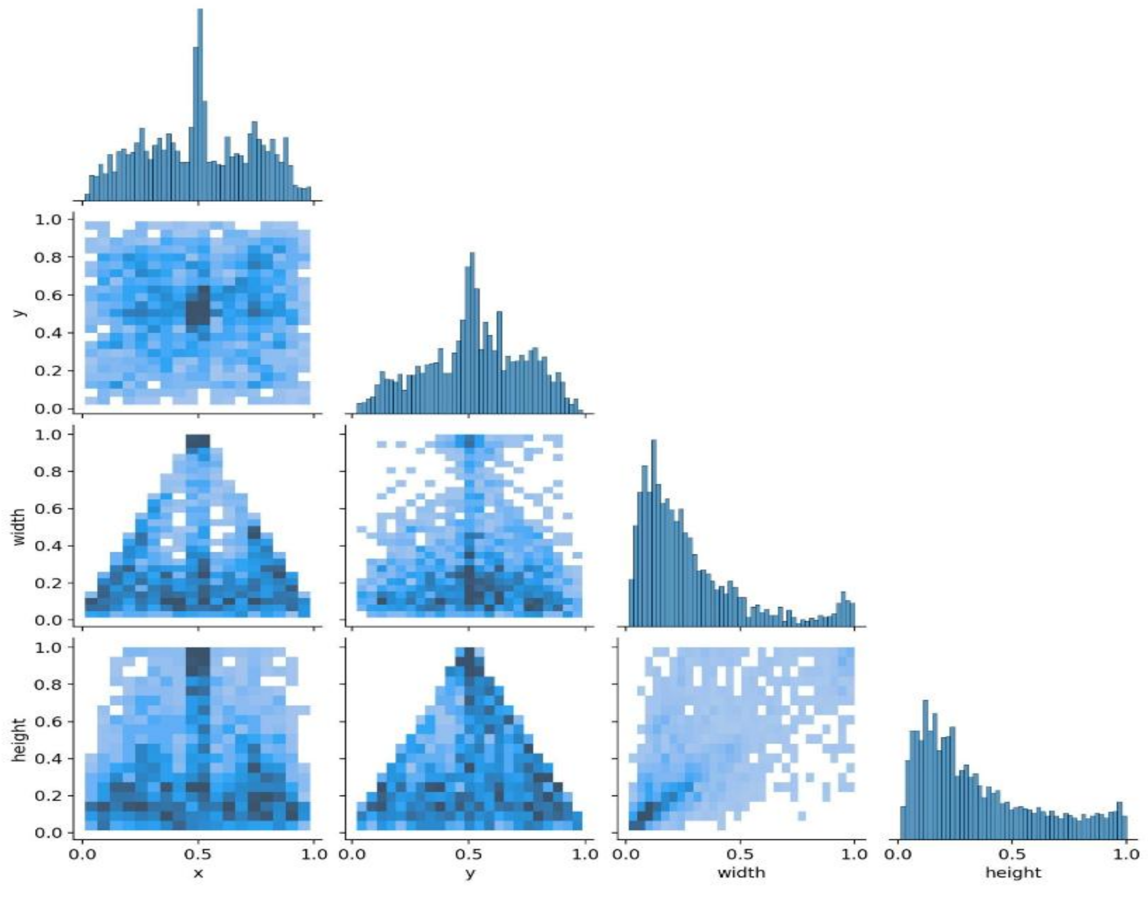
4.3Comparison between Existing and proposed System

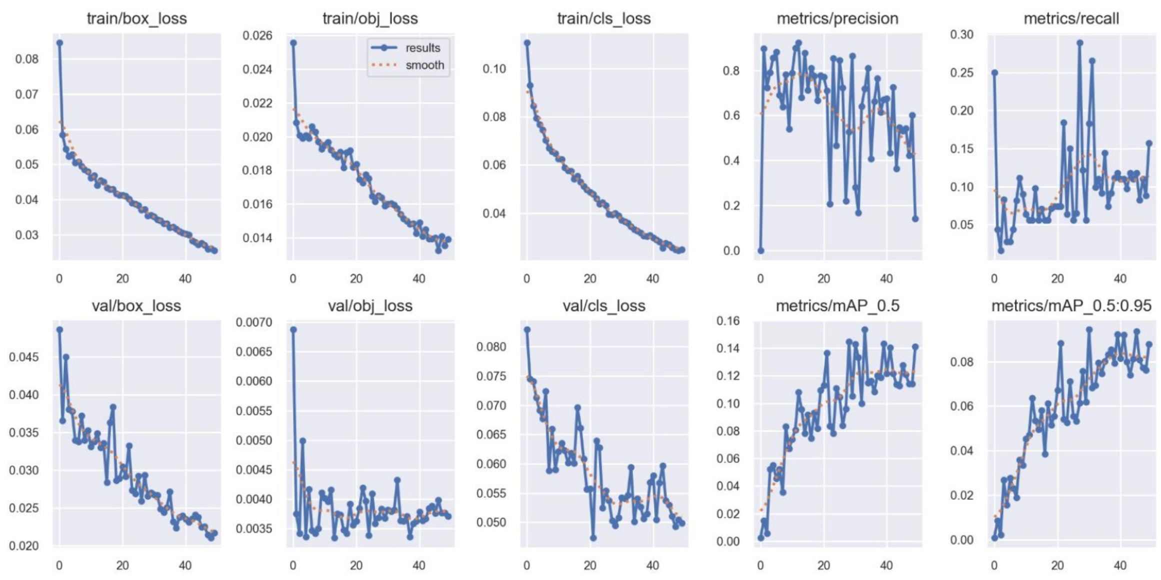
The traditional shopping experience has remained largely unchanged for decades. Shoppers manually pick items from shelves, navigate through crowded aisles, rely on printed or mental shopping lists, and stand in line for manual checkouts. While functional, this system can be time-consuming, inefficient, and error-prone. The proposed Smart Shopping Cart system, driven by machine learning, IoT, and automation, seeks to address these inefficiencies and offer a more seamless and personalized shopping experience. This essay compares the existing and proposed systems across key dimensions.

Existing System: In a traditional shopping environment, customers enter a store with a general idea or a written list of what they need. They navigate the store manually, searching for products, often losing time wandering through aisles. If they forget something, they must retrace their steps. After selecting items, they proceed to a cashier for manual billing and payment, which can involve waiting in line for extended periods. Checkout in a traditional system is a manual process. Shoppers gather all their items and head to a cashier who scans each product, generates a bill, and processes the payment. This is often a time-consuming process, particularly during peak hours. The checkout line can be stressful, and errors in scanning or billing occasionally occur, leading to additional delays. In the existing setup, customers receive little to no personalized product suggestions. While some stores offer loyalty programs that generate occasional personalized discounts or coupons, real-time product recommendations are rare. Customers rely on advertisements, product placements, or their own knowledge to decide what to buy.

Proposed System: The smart shopping cart system transforms this process into a guided, efficient experience. Upon entering the store, the shopper can input the purpose of their visit or allow the system to automatically suggest products based on the items added to the cart. The smart cart assists with creating a dynamic shopping list, suggesting items that complement previous selections. The checkout process is fully automated, with the cart generating a real-time bill and offering contactless payment options, eliminating the need for manual cashier intervention. The smart shopping cart automatically tracks every item placed inside. It scans the products, updates the bill in real-time, and can even suggest last-minute items that complement the user’s shopping history. Once the user is done, they can directly pay via mobile or take the bill to a counter for payment. This reduces waiting time significantly and provides a smooth, hassle-free checkout experience. The smart shopping cart uses machine learning algorithms to analyze the customer's shopping behavior and suggest relevant products. If the user is making chicken biryani and adds chicken, rice, and curd to the cart, the system will suggest spices or other relevant items. It can also tailor recommendations based on dietary preferences, past purchases, and seasonal promotions.

4.4 Results





train/box_loss
train/obj_loss
train/cls_loss
metrics/precision
metrics/recall
val/box_loss
val/obj_loss
val/cls_loss
metrics/mAP_0.5
metrics/mAP_0.5:0.95
results
smooth

Chapter 5

CONCLUSION

The Smart Shopping Cart system represents a transformative leap in the retail shopping experience, combining the power of automation, machine learning, and IoT to create a more efficient and personalized shopping journey. By streamlining tasks such as product selection, shopping list management, and checkout processes, the system drastically reduces time, effort, and human errors. It enhances customer satisfaction through real-time suggestions and personalized experiences, while also enabling retailers to optimize inventory management and reduce operational costs.

The proposed system not only addresses the inefficiencies of the traditional shopping model but also opens up new opportunities for innovation in the retail space. With its ability to scale and adapt to different store sizes, the Smart Shopping Cart is well-positioned to redefine the way we shop, making the process faster, smarter, and more enjoyable. Ultimately, the system paves the way for a future of retail that is both customer-centric and operationally efficient.

In comparing the existing traditional shopping system to the proposed Smart Shopping Cart system, it is evident that the latter offers numerous advantages. The proposed system streamlines the shopping process, provides real-time personalized recommendations, automates billing, and enhances inventory management. It addresses many of the inefficiencies present in the current system, ultimately leading to a more efficient, accurate, and customer-centric retail experience. While the traditional system has served its purpose for decades, the integration of smart technologies into shopping promises to usher in a new era of convenience and operational efficiency for both retailers and shoppers alike.

Chapter 6

FUTURE ENHANCEMENT

In our upcoming research, we intend to improve Our

1) Camera
2) Touchscreen Display
3) Power Supply
4) Networking Components

Chapter 7

SOURCE CODE

```
from flask import Flask, render_template, Response, jsonify
import cv2
import numpy as np

app = Flask(_name_)

# Paths to the YOLO model files
weights_path = "./models/yolov3.weights"
config_path = "./models/yolov3.cfg"
names_path = "./models/coco.names"

# List of classes to ignore (e.g., person)
ignored_classes = {"person"}

# Example product database with details and associations
product_database = {

    "apple": {"price": 7.00, "description": "Fresh apple", "stock": 100, "related": ["banana",
"orange"]},
    "umbrella": {"price": 10.00, "description": "umbrella", "stock": 100, "related": ["rain
coat"]},
    "sports ball": {"price": 20.00, "description": "ball", "stock": 100, "related": ["gloves",
"bat"]},
    "bottle": {"price": 7.00, "description": "water bottle", "stock": 100, "related": ["cup",
"glass"]},
    "lens sollution": {"price": 7.00, "description": "glasses", "stock": 100, "related": ["lens",
"eyewear"]},
    "banana": {"price": 0.50, "description": "Ripe banana", "stock": 150, "related": ["apple",
"orange"]},
    "orange": {"price": 0.75, "description": "Juicy orange", "stock": 200, "related": ["apple",
"banana"]},
    "pen": {"price": 2.00, "description": "Ballpoint pen", "stock": 500, "related":
["notebook", "eraser"]},
    "notebook": {"price": 3.00, "description": "Spiral notebook", "stock": 300, "related":
["pen", "eraser"]},
    "eraser": {"price": 1.00, "description": "White eraser", "stock": 400, "related": ["pen",
"notebook"]},
    "laptop": {"price": 1000.00, "description": "High-performance laptop", "stock": 50,
"related": ["mouse", "keyboard"]},
    "mouse": {"price": 20.00, "description": "Wireless mouse", "stock": 100, "related":
["laptop", "keyboard"]},
    "keyboard": {"price": 30.00, "description": "Mechanical keyboard", "stock": 75,
"related": ["laptop", "mouse"]},
    "shirt": {"price": 15.00, "description": "Casual shirt", "stock": 200, "related": ["jeans",
"shoes"]},
    "jeans": {"price": 25.00, "description": "Blue jeans", "stock": 150, "related": ["shirt",
"shoes"]},
    "shoes": {"price": 50.00, "description": "Running shoes", "stock": 100, "related": ["shirt",
"jeans"]},
    "car": {"price": 250000000.00, "description": "Porsche car", "stock": 3, "related":
["wheels", "indoor spray"]},
    "wheels": {"price": 1000.00, "description": "Alloy wheels", "stock": 10, "related": ["car",
"indoor spray"]},
    "indoor spray": {"price": 5.00, "description": "Air freshener spray", "stock": 100,
"related": ["car", "wheels"]},
    "motorcycle": {"price": 1000000.00, "description": "Ducati motorcycle", "stock": 5,
"related": ["gloves", "glasses"]},
    "gloves": {"price": 50.00, "description": "Leather gloves", "stock": 20, "related":
["motorcycle", "glasses"]},
    "glasses": {"price": 30.00, "description": "Sunglasses", "stock": 50, "related":
["motorcycle", "gloves"]},
    "bicycle": {"price": 500.00, "description": "Mountain bike", "stock": 10, "related": []},
    "chicken": {"price": 10.00, "description": "Fresh chicken", "stock": 50, "related":
["biryani masala", "curd", "spices"]},
    "curd": {"price": 3.00, "description": "Natural curd", "stock": 100, "related": ["biryani
masala", "chicken", "spices"]},
    "spices": {"price": 2.50, "description": "Mixed spices", "stock": 300, "related": ["biryani
masala", "chicken", "curd"]},
    "biryani masala": {"price": 5.00, "description": "Biryani masala spice mix", "stock": 200,
"related": ["chicken", "curd", "spices"]},
    "tv": {"price": 500.00, "description": "Smart TV", "stock": 10, "related": ["remote",
"soundbar"]},
    "remote": {"price": 10.00, "description": "TV remote control", "stock": 50, "related":
["tv", "soundbar"]},
    "soundbar": {"price": 100.00, "description": "Wireless soundbar", "stock": 20, "related":
["tv", "remote"]},
    "sofa": {"price": 1000.00, "description": "comfortable sofa", "stock": 20, "related": ["tv",
"remote"]},
    "cell phone": {"price": 100.00, "description": "Smartphone", "stock": 20, "related":
["laptop"]} ,
    "toothbrush": {"price": 70.00, "description": "toiletries", "stock": 20, "related":
["toothpaste"]},
}

# Load YOLO model
net = cv2.dnn.readNet(weights_path, config_path)
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

# Load the COCO class names
with open(names_path, "r") as f:
    classes = [line.strip() for line in f.readlines()]
```

```
purchased_items = []
detected_items = []  # List to keep track of detected items

def detect_objects(image):
    height, width, _ = image.shape
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    class_ids = []
    confidences = []
    boxes = []
    object_positions = {}

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                label = str(classes[class_id])

                if label not in ignored_classes:
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)
                    object_positions[label] = center_x

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence = confidences[i]
            color = (0, 255, 0)
            cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
            cv2.putText(image, f"{label} {confidence:.2f}", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

```
    return object_positions, image

def add_to_bill(item_name):
    if item_name in product_database:
        purchased_items.append(item_name)
        product_database[item_name]['stock'] -= 1

def detect_movement(prev_positions, curr_positions, threshold=50):
    for item, curr_pos_x in curr_positions.items():
        if item in prev_positions:
            prev_pos_x = prev_positions[item]
            distance = curr_pos_x - prev_pos_x
            if distance > threshold:
                return item
    return None

def gen_frames():
    global detected_items
    cap = cv2.VideoCapture(0)
    prev_positions = {}
    already_detected = set()

    while True:
        success, frame = cap.read()
        if not success:
            break
        else:
            curr_positions, frame = detect_objects(frame)
            moving_item = detect_movement(prev_positions, curr_positions)

            if moving_item and moving_item not in already_detected:
                print(f"Detected: {moving_item}")  # Debug print statement
                add_to_bill(moving_item)
                detected_items.append(moving_item)
                already_detected.add(moving_item)

            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()

            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
        prev_positions = curr_positions

    cap.release()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video_feed')
def video_feed():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/bill')
def bill():
    total = 0
    purchased_summary = []
    for item in purchased_items:
        price = product_database[item]["price"]
        total += price
        purchased_summary.append(f"{item}: ${price}")

    return render_template('bill.html', purchased_summary=purchased_summary, total=total)

@app.route('/get_detected_items')
def get_detected_items():
    global detected_items
    return jsonify({"items": detected_items})

# Paths to the YOLO model files
weights_path = "./models/yolov3.weights"
config_path = "./models/yolov3.cfg"
names_path = "./models/coco.names"

# List of classes to ignore (e.g., person)
ignored_classes = {"person"}

# Example product database with details and associations
product_database = {

    "apple": {"price": 7.00, "description": "Fresh apple", "stock": 100, "related": ["banana",
"orange"]},
    "umbrella": {"price": 10.00, "description": "umbrella", "stock": 100, "related": ["rain
coat"]},
    "sports ball": {"price": 20.00, "description": "ball", "stock": 100, "related": ["gloves",
"bat"]},
    "bottle": {"price": 7.00, "description": "water bottle", "stock": 100, "related": ["cup",
"glass"]},
    "lens sollution": {"price": 7.00, "description": "glasses", "stock": 100, "related": ["lens",
"eyewear"]},
    "banana": {"price": 0.50, "description": "Ripe banana", "stock": 150, "related": ["apple",
"orange"]},
    "orange": {"price": 0.75, "description": "Juicy orange", "stock": 200, "related": ["apple",
"banana"]},
    "pen": {"price": 2.00, "description": "Ballpoint pen", "stock": 500, "related":
["notebook", "eraser"]},
    "notebook": {"price": 3.00, "description": "Spiral notebook", "stock": 300, "related":
["pen", "eraser"]},
    "eraser": {"price": 1.00, "description": "White eraser", "stock": 400, "related": ["pen",
"notebook"]},
    "laptop": {"price": 1000.00, "description": "High-performance laptop", "stock": 50,
"related": ["mouse", "keyboard"]},
    "mouse": {"price": 20.00, "description": "Wireless mouse", "stock": 100, "related":
["laptop", "keyboard"]},
    "keyboard": {"price": 30.00, "description": "Mechanical keyboard", "stock": 75,
"related": ["laptop", "mouse"]},
    "shirt": {"price": 15.00, "description": "Casual shirt", "stock": 200, "related": ["jeans",
"shoes"]},
    "jeans": {"price": 25.00, "description": "Blue jeans", "stock": 150, "related": ["shirt",
"shoes"]},
    "shoes": {"price": 50.00, "description": "Running shoes", "stock": 100, "related": ["shirt",
"jeans"]},
    "car": {"price": 250000000.00, "description": "Porsche car", "stock": 3, "related":
["wheels", "indoor spray"]},
    "wheels": {"price": 1000.00, "description": "Alloy wheels", "stock": 10, "related": ["car",
"indoor spray"]},
    "indoor spray": {"price": 5.00, "description": "Air freshener spray", "stock": 100,
"related": ["car", "wheels"]},
    "motorcycle": {"price": 1000000.00, "description": "Ducati motorcycle", "stock": 5,
"related": ["gloves", "glasses"]},
    "gloves": {"price": 50.00, "description": "Leather gloves", "stock": 20, "related":
["motorcycle", "glasses"]},
    "glasses": {"price": 30.00, "description": "Sunglasses", "stock": 50, "related":
["motorcycle", "gloves"]},
    "bicycle": {"price": 500.00, "description": "Mountain bike", "stock": 10, "related": []},
    "chicken": {"price": 10.00, "description": "Fresh chicken", "stock": 50, "related":
["biryani masala", "curd", "spices"]},
    "curd": {"price": 3.00, "description": "Natural curd", "stock": 100, "related": ["biryani
masala", "chicken", "spices"]},
    "spices": {"price": 2.50, "description": "Mixed spices", "stock": 300, "related": ["biryani
masala", "chicken", "curd"]},
    "biryani masala": {"price": 5.00, "description": "Biryani masala spice mix", "stock": 200,
"related": ["chicken", "curd", "spices"]},
    "tv": {"price": 500.00, "description": "Smart TV", "stock": 10, "related": ["remote",
"soundbar"]},
    "remote": {"price": 10.00, "description": "TV remote control", "stock": 50, "related":
["tv", "soundbar"]},
    "soundbar": {"price": 100.00, "description": "Wireless soundbar", "stock": 20, "related":
["tv", "remote"]},
    "sofa": {"price": 1000.00, "description": "comfortable sofa", "stock": 20, "related": ["tv",
"remote"]},
    "cell phone": {"price": 100.00, "description": "Smartphone", "stock": 20, "related":
["laptop"]} ,
    "toothbrush": {"price": 70.00, "description": "toiletries", "stock": 20, "related":
["toothpaste"]},
}

# Load YOLO model
net = cv2.dnn.readNet(weights_path, config_path)
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

# Load the COCO class names
with open(names_path, "r") as f:
    classes = [line.strip() for line in f.readlines()]

purchased_items = []
detected_items = []  # List to keep track of detected items

def detect_objects(image):
    height, width, _ = image.shape
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    class_ids = []
    confidences = []
    boxes = []
    object_positions = {}

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                label = str(classes[class_id])

                if label not in ignored_classes:
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)
                    object_positions[label] = center_x
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence = confidences[i]
            color = (0, 255, 0)
            cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
            cv2.putText(image, f"{label} {confidence:.2f}", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    return object_positions, image

def add_to_bill(item_name):
    if item_name in product_database:
        purchased_items.append(item_name)
        product_database[item_name]['stock'] -= 1

def detect_movement(prev_positions, curr_positions, threshold=50):
    for item, curr_pos_x in curr_positions.items():
        if item in prev_positions:
            prev_pos_x = prev_positions[item]
            distance = curr_pos_x - prev_pos_x
            if distance > threshold:
                return item
    return None

# Paths to the YOLO model files
weights_path = "./models/yolov3.weights"
config_path = "./models/yolov3.cfg"
names_path = "./models/coco.names"

# List of classes to ignore (e.g., person)
ignored_classes = {"person"}

# Example product database with details and associations
product_database = {

    "apple": {"price": 7.00, "description": "Fresh apple", "stock": 100, "related": ["banana",
"orange"]},
    "umbrella": {"price": 10.00, "description": "umbrella", "stock": 100, "related": ["rain
coat"]},
    "sports ball": {"price": 20.00, "description": "ball", "stock": 100, "related": ["gloves",
"bat"]},
    "bottle": {"price": 7.00, "description": "water bottle", "stock": 100, "related": ["cup",
"glass"]},
    "lens sollution": {"price": 7.00, "description": "glasses", "stock": 100, "related": ["lens", "eyewear"]},
    "banana": {"price": 0.50, "description": "Ripe banana", "stock": 150, "related": ["apple", "orange"]},
    "orange": {"price": 0.75, "description": "Juicy orange", "stock": 200, "related": ["apple", "banana"]},
    "pen": {"price": 2.00, "description": "Ballpoint pen", "stock": 500, "related": ["notebook", "eraser"]},
    "notebook": {"price": 3.00, "description": "Spiral notebook", "stock": 300, "related": ["pen", "eraser"]},
    "eraser": {"price": 1.00, "description": "White eraser", "stock": 400, "related": ["pen", "notebook"]},
    "laptop": {"price": 1000.00, "description": "High-performance laptop", "stock": 50, "related": ["mouse", "keyboard"]},
    "mouse": {"price": 20.00, "description": "Wireless mouse", "stock": 100, "related": ["laptop", "keyboard"]},
    "keyboard": {"price": 30.00, "description": "Mechanical keyboard", "stock": 75, "related": ["laptop", "mouse"]},
    "shirt": {"price": 15.00, "description": "Casual shirt", "stock": 200, "related": ["jeans", "shoes"]},
    "jeans": {"price": 25.00, "description": "Blue jeans", "stock": 150, "related": ["shirt", "shoes"]},
    "shoes": {"price": 50.00, "description": "Running shoes", "stock": 100, "related": ["shirt", "jeans"]},
    "car": {"price": 250000000.00, "description": "Porsche car", "stock": 3, "related": ["wheels", "indoor spray"]},
    "wheels": {"price": 1000.00, "description": "Alloy wheels", "stock": 10, "related": ["car", "indoor spray"]},
    "indoor spray": {"price": 5.00, "description": "Air freshener spray", "stock": 100, "related": ["car", "wheels"]},
    "motorcycle": {"price": 1000000.00, "description": "Ducati motorcycle", "stock": 5, "related": ["gloves", "glasses"]},
    "gloves": {"price": 50.00, "description": "Leather gloves", "stock": 20, "related": ["motorcycle", "glasses"]},
    "glasses": {"price": 30.00, "description": "Sunglasses", "stock": 50, "related": ["motorcycle", "gloves"]},
    "bicycle": {"price": 500.00, "description": "Mountain bike", "stock": 10, "related": []},
    "chicken": {"price": 10.00, "description": "Fresh chicken", "stock": 50, "related": ["biryani masala", "curd", "spices"]},
    "curd": {"price": 3.00, "description": "Natural curd", "stock": 100, "related": ["biryani masala", "chicken", "spices"]},
    "spices": {"price": 2.50, "description": "Mixed spices", "stock": 300, "related": ["biryani masala", "chicken", "curd"]},
    "biryani masala": {"price": 5.00, "description": "Biryani masala spice mix", "stock": 200, "related": ["chicken", "curd", "spices"]},
    "tv": {"price": 500.00, "description": "Smart TV", "stock": 10, "related": ["remote", "soundbar"]},
    "remote": {"price": 10.00, "description": "TV remote control", "stock": 50, "related": ["tv", "soundbar"]},
    "soundbar": {"price": 100.00, "description": "Wireless soundbar", "stock": 20, "related":
["tv", "remote"]},
    "sofa": {"price": 1000.00, "description": "comfortable sofa", "stock": 20, "related": ["tv",
"remote"]},
    "cell phone": {"price": 100.00, "description": "Smartphone", "stock": 20, "related":
["laptop"]} ,
    "toothbrush": {"price": 70.00, "description": "toiletries", "stock": 20, "related":
    ["toothpaste"]},
    }

    # Load YOLO model
    net = cv2.dnn.readNet(weights_path, config_path)
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

    # Load the COCO class names
    with open(names_path, "r") as f:
    classes = [line.strip() for line in f.readlines()]

    purchased_items = []
    detected_items = []  # List to keep track of detected items

    def detect_objects(image):
    height, width, _ = image.shape
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    class_ids = []
    confidences = []
    boxes = []
    object_positions = {}

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                label = str(classes[class_id])
                if label not in ignored_classes:
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)
                    object_positions[label] = center_x

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence = confidences[i]
            color = (0, 255, 0)
            cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
            cv2.putText(image, f"{label} {confidence:.2f}", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    return object_positions, image

def add_to_bill(item_name):
    if item_name in product_database:
        purchased_items.append(item_name)
        product_database[item_name]['stock'] -= 1

def detect_movement(prev_positions, curr_positions, threshold=50):
    for item, curr_pos_x in curr_positions.items():
        if item in prev_positions:
            prev_pos_x = prev_positions[item]
            distance = curr_pos_x - prev_pos_x
            if distance > threshold:
                return item
    return None

# Paths to the YOLO model files
weights_path = "./models/yolov3.weights"
config_path = "./models/yolov3.cfg"
names_path = "./models/coco.names"

# List of classes to ignore (e.g., person)
ignored_classes = {"person"}

# Example product database with details and associations
product_database = {

    "apple": {"price": 7.00, "description": "Fresh apple", "stock": 100, "related": ["banana",
"orange"]},
    "umbrella": {"price": 10.00, "description": "umbrella", "stock": 100, "related": ["rain
coat"]},
    "sports ball": {"price": 20.00, "description": "ball", "stock": 100, "related": ["gloves",
"bat"]},
    "bottle": {"price": 7.00, "description": "water bottle", "stock": 100, "related": ["cup",
"glass"]},
    "lens sollution": {"price": 7.00, "description": "glasses", "stock": 100, "related": ["lens",
"eyewear"]},
    "banana": {"price": 0.50, "description": "Ripe banana", "stock": 150, "related": ["apple",
"orange"]},
    "orange": {"price": 0.75, "description": "Juicy orange", "stock": 200, "related": ["apple",
"banana"]},
    "pen": {"price": 2.00, "description": "Ballpoint pen", "stock": 500, "related":
["notebook", "eraser"]},
    "notebook": {"price": 3.00, "description": "Spiral notebook", "stock": 300, "related":
["pen", "eraser"]},
    "eraser": {"price": 1.00, "description": "White eraser", "stock": 400, "related": ["pen",
"notebook"]},
    "laptop": {"price": 1000.00, "description": "High-performance laptop", "stock": 50,
"related": ["mouse", "keyboard"]},
    "mouse": {"price": 20.00, "description": "Wireless mouse", "stock": 100, "related":
["laptop", "keyboard"]},
    "keyboard": {"price": 30.00, "description": "Mechanical keyboard", "stock": 75,
"related": ["laptop", "mouse"]},
    "shirt": {"price": 15.00, "description": "Casual shirt", "stock": 200, "related": ["jeans",
"shoes"]},
    "jeans": {"price": 25.00, "description": "Blue jeans", "stock": 150, "related": ["shirt",
"shoes"]},
    "shoes": {"price": 50.00, "description": "Running shoes", "stock": 100, "related": ["shirt",
"jeans"]},
    "car": {"price": 250000000.00, "description": "Porsche car", "stock": 3, "related":
["wheels", "indoor spray"]},
    "wheels": {"price": 1000.00, "description": "Alloy wheels", "stock": 10, "related": ["car",
"indoor spray"]},
    "indoor spray": {"price": 5.00, "description": "Air freshener spray", "stock": 100,
"related": ["car", "wheels"]},
    "motorcycle": {"price": 1000000.00, "description": "Ducati motorcycle", "stock": 5,
"related": ["gloves", "glasses"]},
    "gloves": {"price": 50.00, "description": "Leather gloves", "stock": 20, "related":
["motorcycle", "glasses"]},
    "glasses": {"price": 30.00, "description": "Sunglasses", "stock": 50, "related":
["motorcycle", "gloves"]},
    "bicycle": {"price": 500.00, "description": "Mountain bike", "stock": 10, "related": []},
    "chicken": {"price": 10.00, "description": "Fresh chicken", "stock": 50, "related":
["biryani masala", "curd", "spices"]},
    "curd": {"price": 3.00, "description": "Natural curd", "stock": 100, "related": ["biryani
masala", "chicken", "spices"]},
    "spices": {"price": 2.50, "description": "Mixed spices", "stock": 300, "related": ["biryani
masala", "chicken", "curd"]},
    "biryani masala": {"price": 5.00, "description": "Biryani masala spice mix", "stock": 200, "related": ["chicken", "curd", "spices"]},
    "tv": {"price": 500.00, "description": "Smart TV", "stock": 10, "related": ["remote", "soundbar"]},
    "remote": {"price": 10.00, "description": "TV remote control", "stock": 50, "related": ["tv", "soundbar"]},
    "soundbar": {"price": 100.00, "description": "Wireless soundbar", "stock": 20, "related": ["tv", "remote"]},
    "sofa": {"price": 1000.00, "description": "comfortable sofa", "stock": 20, "related": ["tv", "remote"]},
    "cell phone": {"price": 100.00, "description": "Smartphone", "stock": 20, "related": ["laptop"]} ,
    "toothbrush": {"price": 70.00, "description": "toiletries", "stock": 20, "related": ["toothpaste"]},
}

# Load YOLO model
net = cv2.dnn.readNet(weights_path, config_path)
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

# Load the COCO class names
with open(names_path, "r") as f:
    classes = [line.strip() for line in f.readlines()]

purchased_items = []
detected_items = []  # List to keep track of detected items

def detect_objects(image):
    height, width, _ = image.shape
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    class_ids = []
    confidences = []
    boxes = []
    object_positions = {}

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                label = str(classes[class_id])

                if label not in ignored_classes:
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)
                    object_positions[label] = center_x

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence = confidences[i]
            color = (0, 255, 0)
            cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
            cv2.putText(image, f"{label} {confidence:.2f}", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    return object_positions, image

def add_to_bill(item_name):
    if item_name in product_database:
        purchased_items.append(item_name)
        product_database[item_name]['stock'] -= 1

def detect_movement(prev_positions, curr_positions, threshold=50):
    for item, curr_pos_x in curr_positions.items():
        if item in prev_positions:
            prev_pos_x = prev_positions[item]
            distance = curr_pos_x - prev_pos_x
            if distance > threshold:
                return item
    return None

if _name_ == "_main_":
    app.run(debug=True)
```

References

S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, June 2017.

A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Communications of the ACM, vol. 60, no. 6, pp. 84-90, 2017.

X. Chen, A. Mishra, N. R. Kejriwal, and B. Zhang, "Smart Trolley: An Intelligent Shopping Cart Using Deep Learning," IEEE Access, vol. 8, pp. 167792-167802, 2020.

P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for YouTube Recommendations," in Proceedings of the 10th ACM Conference on Recommender Systems, 2016, pp. 191-198.

A. Toshev and C. Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1653-1660.

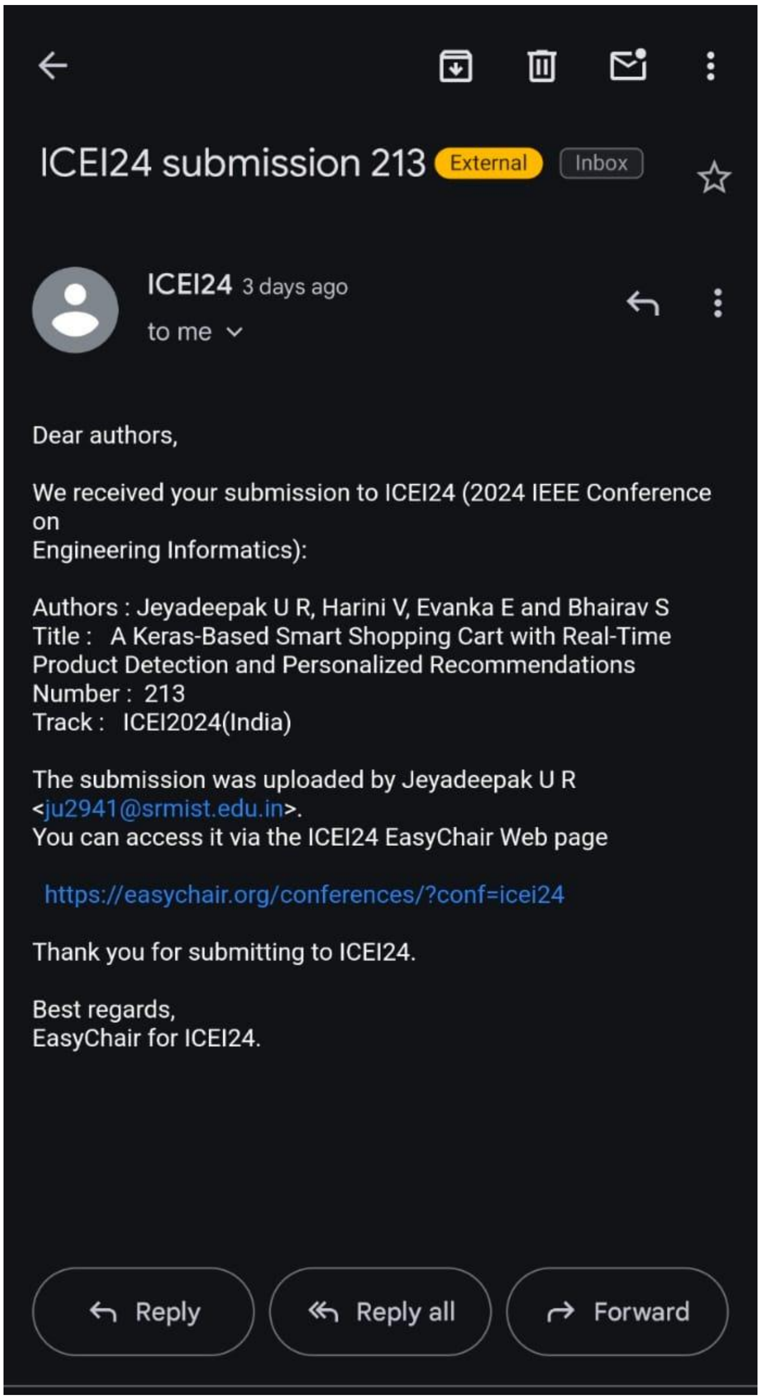
J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv preprint arXiv:1804.02767, 2018.

Y. He, Y. Cao, J. Feng, and X. Chu, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 784-800.

Y. Li, J. Zhang, and Y. Zhu, "A Personalized Recommendation System Based on Collaborative Filtering Algorithm," IEEE Access, vol. 7, pp. 17250-17257, 2019.

M. N. Kim, "Design of a Smart Shopping Cart System Using an RFID Sensor and Real-Time Data Processing," IEEE Sensors Journal, vol. 19, no. 24, pp. 12392-12397, Dec. 2019.

PUBLICATION PROOF


ICEI24 submission 213

ICEI24 3 days ago

to me

Dear authors,

We received your submission to ICEI24 (2024 IEEE Conference on
Engineering Informatics):

Authors : Jeyadeepak U R, Harini V, Evanka E and Bhairav S
Title : A Keras-Based Smart Shopping Cart with Real-Time Product Detection and Personalized Recommendations
Number : 213
Track : ICEI2024(India)

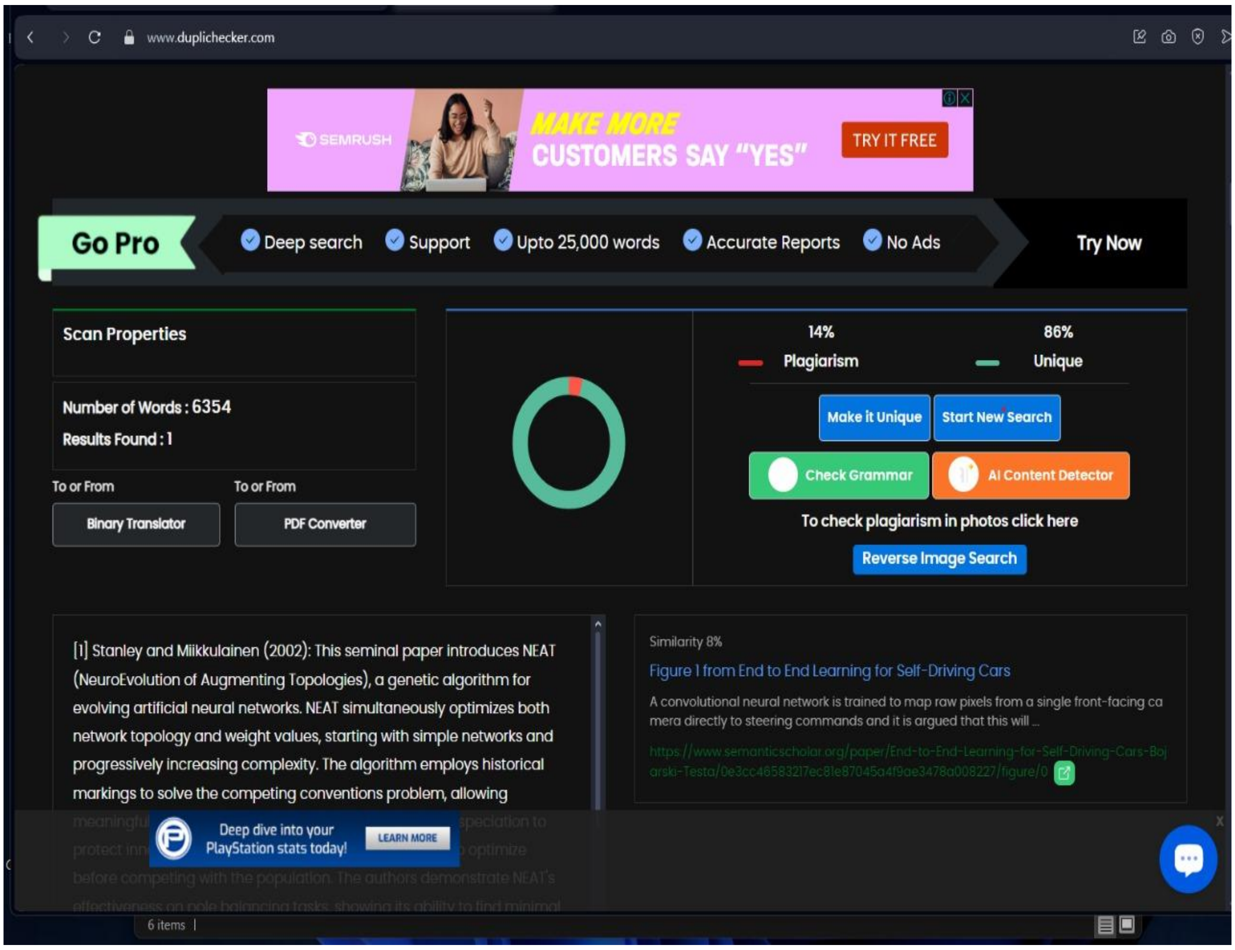
The submission was uploaded by Jeyadeepak U R
<ju2941@srmist.edu.in>.
You can access it via the ICEI24 EasyChair Web page

https://easychair.org/conferences/?conf=icei24

Thank you for submitting to ICEI24.

Best regards,
EasyChair for ICEI24.

Plagiarism checker



Scan Properties

Number of Words : 6354
Results Found : 1

14% Plagiarism
86% Unique

To check plagiarism in photos click here

[1] Stanley and Miikkulainen (2002): This seminal paper introduces NEAT (NeuroEvolution of Augmenting Topologies), a genetic algorithm for evolving artificial neural networks. NEAT simultaneously optimizes both network topology and weight values, starting with simple networks and progressively increasing complexity. The algorithm employs historical markings to solve the competing conventions problem, allowing meaningful … speciation to protect inn… optimize before competing with the population. The authors demonstrate NEAT's effectiveness on pole balancing tasks, showing its ability to find minimal

Similarity 8%
Figure 1 from End to End Learning for Self-Driving Cars
A convolutional neural network is trained to map raw pixels from a single front-facing ca mera directly to steering commands and it is argued that this will ...
https://www.semanticscholar.org/paper/End-to-End-Learning-for-Self-Driving-Cars-Bojarski-Testa/0e3cc46583217ec81e87045a4f9ae3478a008227/figure/0