Industry 4.0 Warehouse Management System

R25-062

Meet our team

Mayukha Palihena

IT21079672

Adilu Salinda

IT21822780

Sidath Tharana

IT21822094

Visura Suranjith

IT21318184

Dr. Samantha Rajapaksha

Supervisor

Dr. Dinuka Wijendra

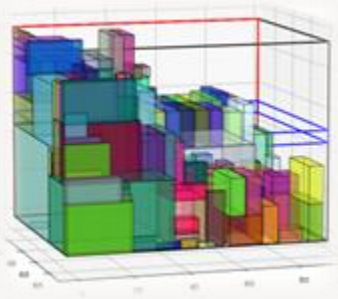
Co-Supervisor

Abstract

The goal of this research is to develop an Industry 4.0-driven Warehouse Management System (WMS) designed to optimize operations, enhance safety, and reduce costs. The system integrates container space optimization, order picking route optimization, real-time fire detection using computer vision, and alerts for abnormal stock movement, all working together to improve warehouse efficiency and safety. The project utilizes AI, computer vision, and real-time data processing to address common warehouse challenges, improving decision-making and operational performance.

Introduction

- **Overview:**
  - Warehouses are vital components of modern supply chains. Optimizing warehouse operations leads to significant cost savings, improved efficiency, and better service delivery.
  - This research focuses on applying Industry 4.0 technologies to improve warehouse management, ensuring the efficient use of space, optimized order picking, better safety measures, and real-time monitoring of stock movements.
- **Challenges Addressed:**
  - Space utilization inefficiencies
  - Time-consuming order picking processes
  - Insufficient fire detection systems
  - Lack of real-time stock movement monitoring



literature survey

- Solutions such as SAP Ex WMS, Oracle WMS exist for warehouse needs, but they lack Industry 4.0 focused features such as Stock Movement Instabilities, Fire Detection with Computer Vision, Order Picking Route Optimization & Container Space Optimization.

ORACLE®
WAREHOUSE MANAGEMENT SYSTEMS

SAP®

Research Gap

The research gap includes,

the lack of dynamic space optimization systems, inefficient order picking, the absence of AI-driven real-time fire risk assessments, and limited predictive analytics for real-time stock movement alerts, all of which hinder warehouse efficiency and safety.

Research Questions

- **Limited Fire Detection:** How can we provide real-time fire alerts near critical areas like racks, hindering rapid response ?
- **Inefficient Space Utilization:** Lack of intelligent systems for dynamically optimizing warehouse space leads to inefficiencies. How can we address that ?
- **Route Inefficiency:** How can we save time and energy by minimizing warehouse route problems ?
- **Stock Instabilities:** How to predict incidents which address sudden movements in stock, leading to undetected inventory issues ?

Main Objective

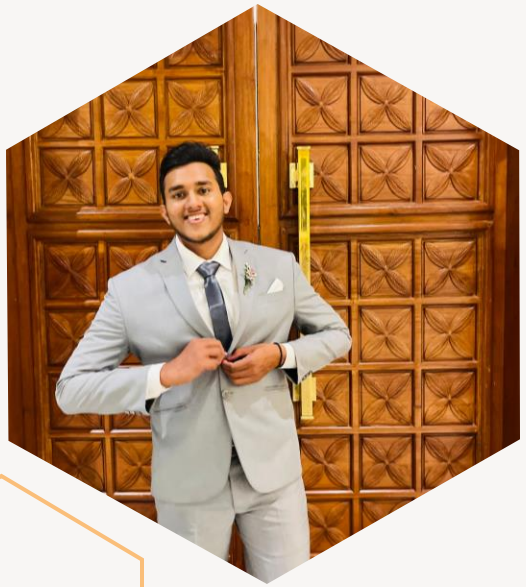
To develop an optimized warehouse management system that enhances space utilization, improves inventory turnover adaptability, streamlines year-end evaluations and mitigates risks associated with sensitive goods

Sub Objectives

- Maximize warehouse space utilization by leveraging algorithms to dynamically allocate inventory.
- Develop a real-time, optimized order picking and forklift path system that reduces time, enhances safety, and minimizes congestion.
- Implementing systems to detect fire hazards based on proximity to critical zones and trigger real-time alerts.
- Analyse and predict for abnormal stock movements, improving safety and operational efficiency.

Aspect 01

Optimizing the arrangement of products within the best location of a warehouse, to maximize space utilization.



Introduction

Why is product arrangement optimization important?

- Efficient use of warehouse space ensures cost savings.
- Proper arrangement reduces retrieval time and operational inefficiencies.
- Aligns with the overall objective of maximizing warehouse CBM utilization.

Problems

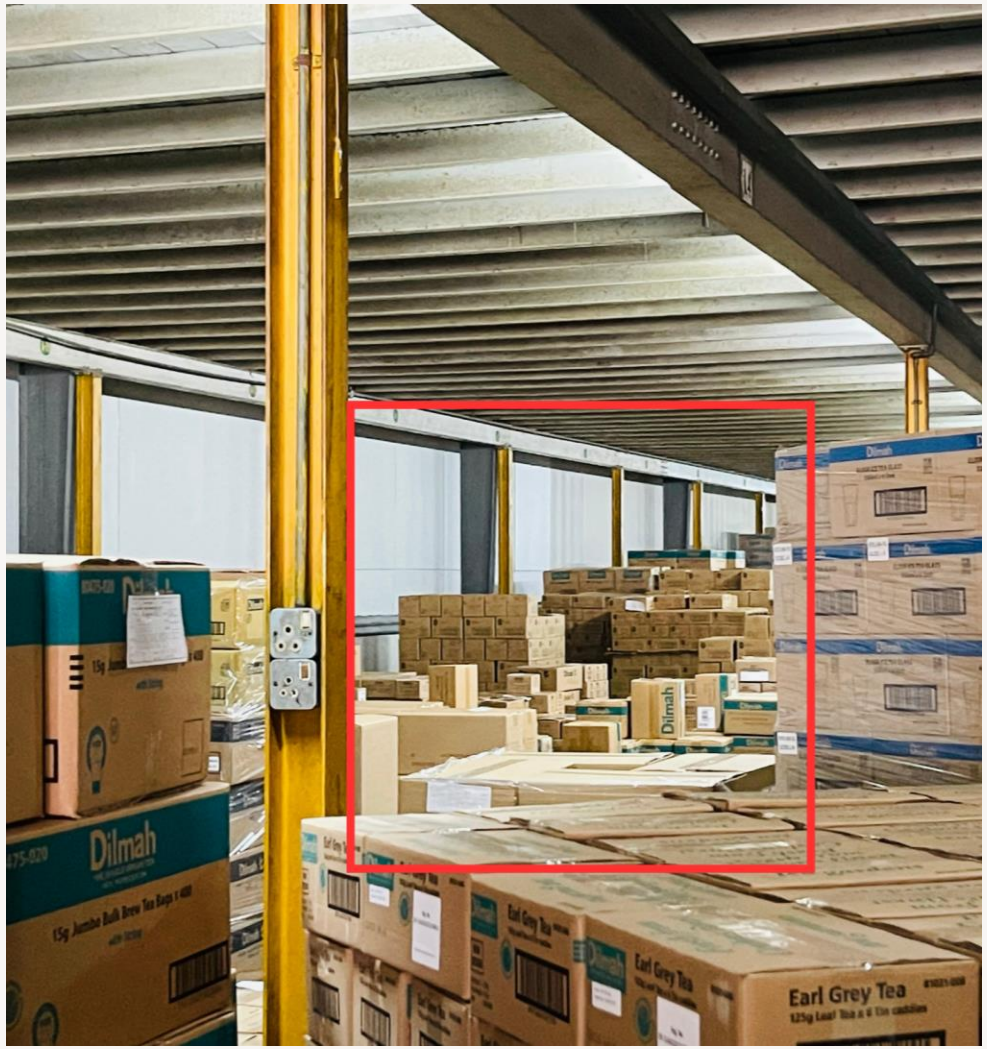
Disorganized layout, leaving gaps and unused areas leads to space wastage

Need for a systematic method to arrange products within predefined locations.

Problems

Difficulty in accommodating products while maintaining accessibility

Designed for horizontal storage, leaving significant vertical space underutilized.



Literature Review

[1] Bartholdi, J.J.; Hackman, S.T. "Warehouse and Distribution Science". 2019. Available online: https://www.warehouse-science.com/book/index.html (accessed on 30 May 2020)

[2] Ignacio Angulo, J.D; Jenny Fajardo, H.R "Optimization of Warehouse Layout for the Minimization of Operation Times". September 2021, Hybrid Artificial Intelligent Systems (pp.649-658)

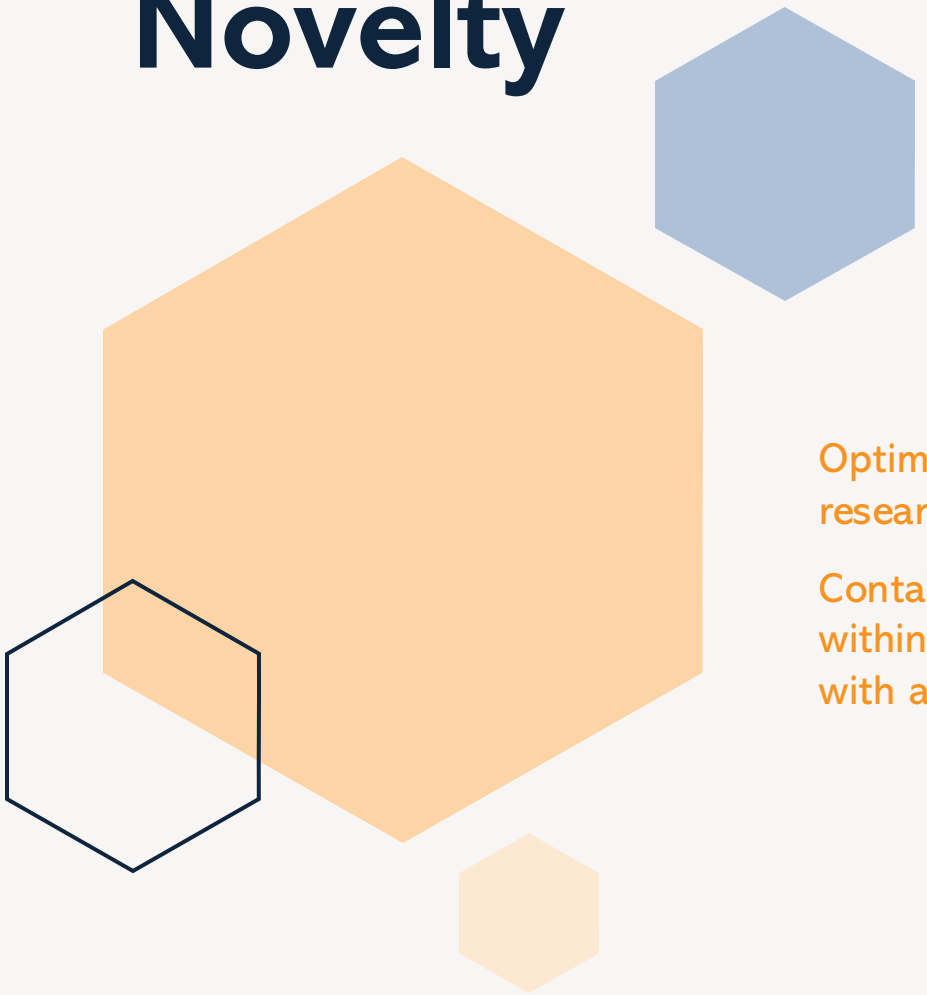
[3] Bartholdi JJ, Hackman S (2008) "Allocating space in a forward pick area of a distribution center for small parts". IIE Trans 40(11):1046–1053. https://doi.org/10.1080/07408170802167662

[4] Ammouri B. 2023. "Digital twin technologies enable facility efficiency. Plant Engineering" 77(I) DOI 10.1016/j.jmsy.2019.10.001.

Research Gap

	Research A [1]	Research B [2]	Research C [3]	Research D [4]	Proposed Solution
Product Arrangement in Location to Maximize CBM Utilization	✗	✗	✗	✗	✓
Real-Time Visualization of Product Arrangement (2D/3D)	✗	✗	✗	✓	✓
Machine Learning and Algorithms for Product Arrangement	✓	✓	✗	✗	✓
Usability and Accessibility in Storing Products	✓	✗	✓	✗	✓

Novelty



Optimizing product arrangements within the best warehouse location, this research leverages CBM-specific strategies, real-time 2D visualization.

Container space optimization: - Leveraging maximum space available within a rack without wasting any space. Drawing the optimized space with allocated objects.

Objectives

Main Objective

Optimizing the arrangement of products within the best location of a warehouse, to maximize space utilization.

Sub Objectives

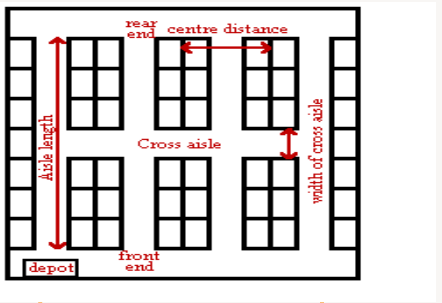
- Create predictive models based on CBM-specific data to optimize product storage arrangements dynamically
- Provide real-time 2D visual representations of optimized product arrangements
- Enhance Usability and Accessibility

Methodology

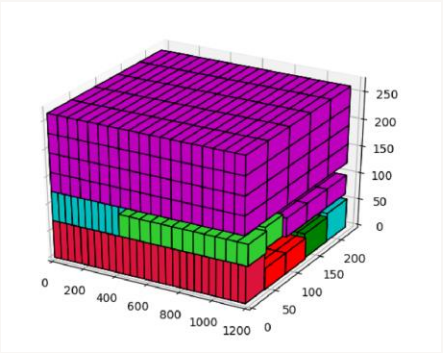
Product information with CBM

Data Analyzation

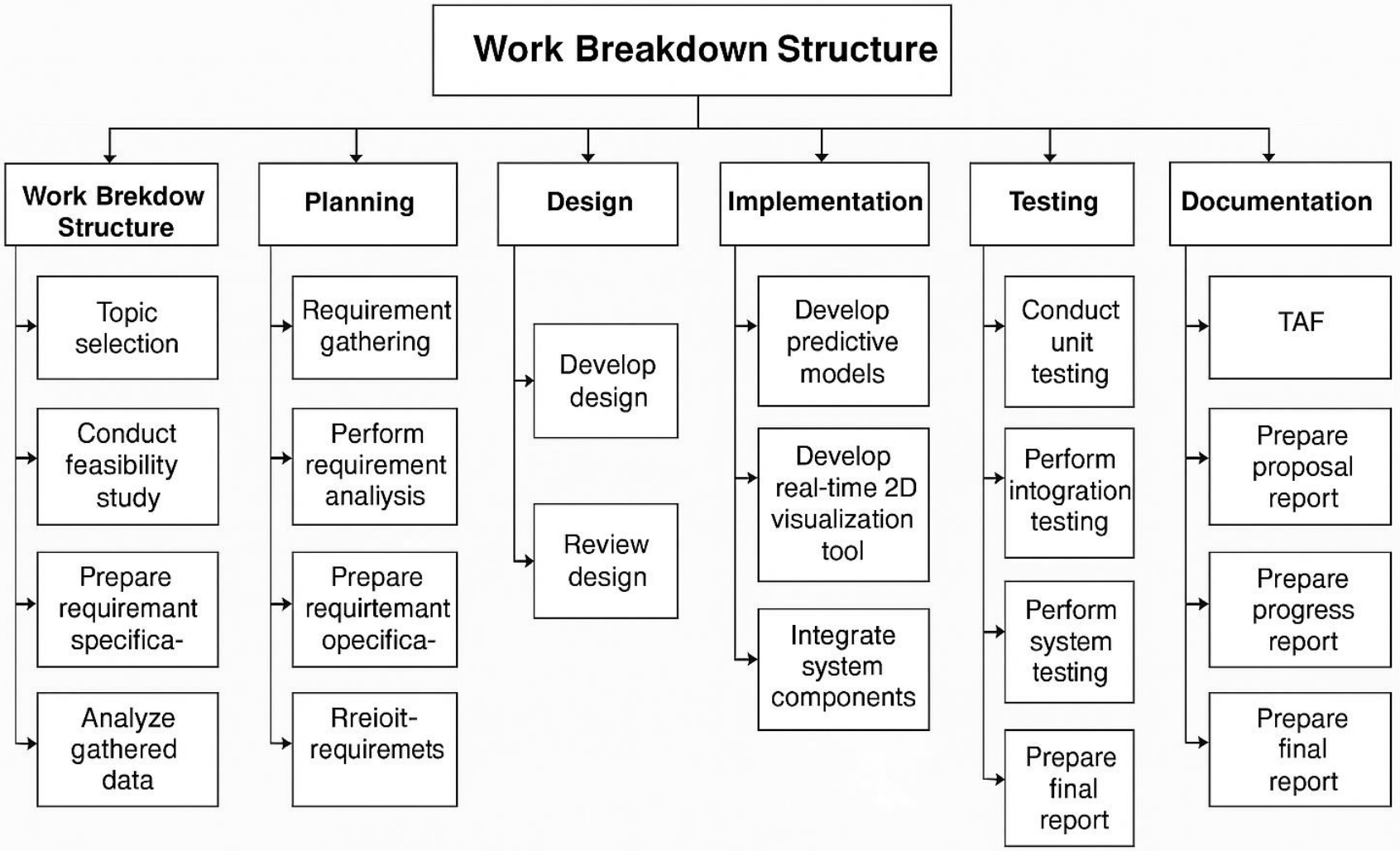
Selecting the best location of the warehouse to store the product



Creating the Arrangement in selected location



Output with Selected location and Arrangement visualization


Work Breakdown Structure
Work Brekdow Structure
Topic selection
Conduct feasibility study
Prepare requiremant specifica-
Analyze gathered data
Planning
Requirement gathering
Perform requirement analiysis
Prepare requirtemant opecifica-
Rreioit-requiremets
Design
Develop design
Review design
Implementation
Develop predictive models
Develop real-time 2D visualization tool
Integrate system components
Testing
Conduct unit testing
Perform intogration testing
Perform system testing
Prepare final report
Documentation
TAF
Prepare proposal report
Prepare progress report
Prepare final report

Progress as Now

Container Space Optimization :

Leveraging maximum space available within the rack without wasting any space

Draw the optimized space with allocated objects.

Tech Stacks :

Python

Matplotlib (For animation and drawing objects)

Customized best fit degree algorithm

```
class FixedShelfPacker:  1 usage
    def __init__(self, shelf_width, shelf_height, shelf_count, compatibility_rules):
        self.fig, self.ax = plt.subplots(figsize=(8, 6))
        self.current_item_index = 0

    def add_item(self, width, height, item_type):  1 usage
        self.items.append((width, height, item_type))


    # searching for available free spaces in the shelves
    def find_best_shelf(self, item_type, width, height):  1 usage
        best_shelf = None
        min_waste = float("inf")

        for shelf in self.shelves:
            if not shelf["compatibility"] or item_type in shelf["compatibility"]:
                for i, (x, y, w, h) in enumerate(shelf["free_spaces"]):
                    if width <= w and height <= h:
                        waste = (w - width) * (h - height)
                        if waste < min_waste:
                            min_waste = waste
                            best_shelf = (shelf, i, x, y, width, height)

        return best_shelf
```

Output

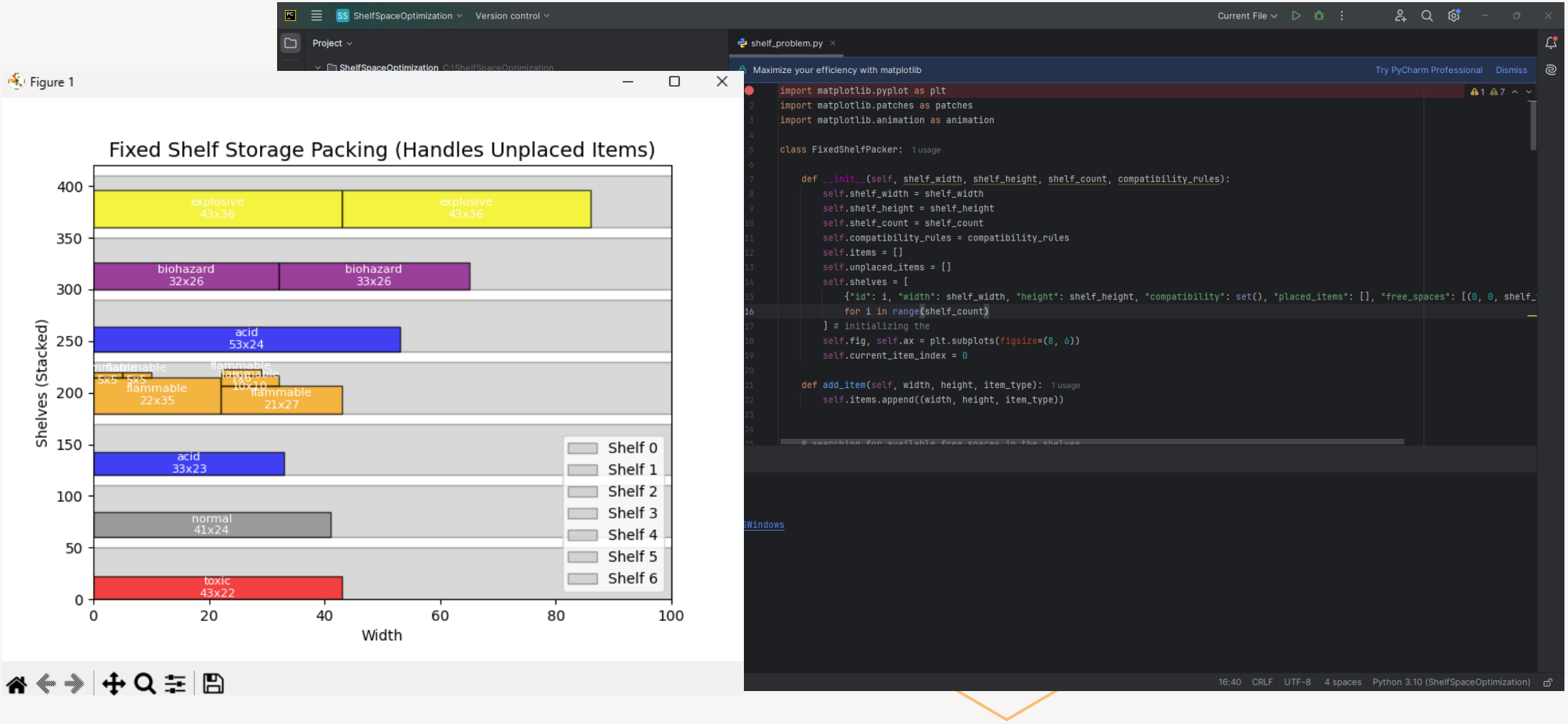
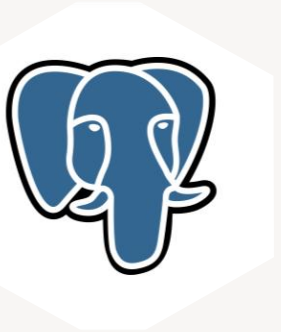


Figure 1
Fixed Shelf Storage Packing (Handles Unplaced Items)
Shelves (Stacked)
Width
explosive 43x36
explosive 43x36
biohazard 32x26
biohazard 33x26
acid 53x24
flammable 22x35
flammable 21x27
acid 33x23
normal 41x24
toxic 43x22
Shelf 0
Shelf 1
Shelf 2
Shelf 3
Shelf 4
Shelf 5
Shelf 6
ShelfSpaceOptimization
Version control
Current File
Project
shelf_problem.py
Maximize your efficiency with matplotlib
Try PyCharm Professional
Dismiss
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.animation as animation
class FixedShelfPacker:
def __init__(self, shelf_width, shelf_height, shelf_count, compatibility_rules):
self.shelf_width = shelf_width
self.shelf_height = shelf_height
self.shelf_count = shelf_count
self.compatibility_rules = compatibility_rules
self.items = []
self.unplaced_items = []
self.shelves = [
{"id": i, "width": shelf_width, "height": shelf_height, "compatibility": set(), "placed_items": [], "free_spaces": [(0, 0, shelf_
for i in range(shelf_count)
] # initializing the
self.fig, self.ax = plt.subplots(figsize=(8, 6))
self.current_item_index = 0
def add_item(self, width, height, item_type):
self.items.append((width, height, item_type))
16:40
CRLF
UTF-8
4 spaces
Python 3.10 (ShelfSpaceOptimization)

Technologies

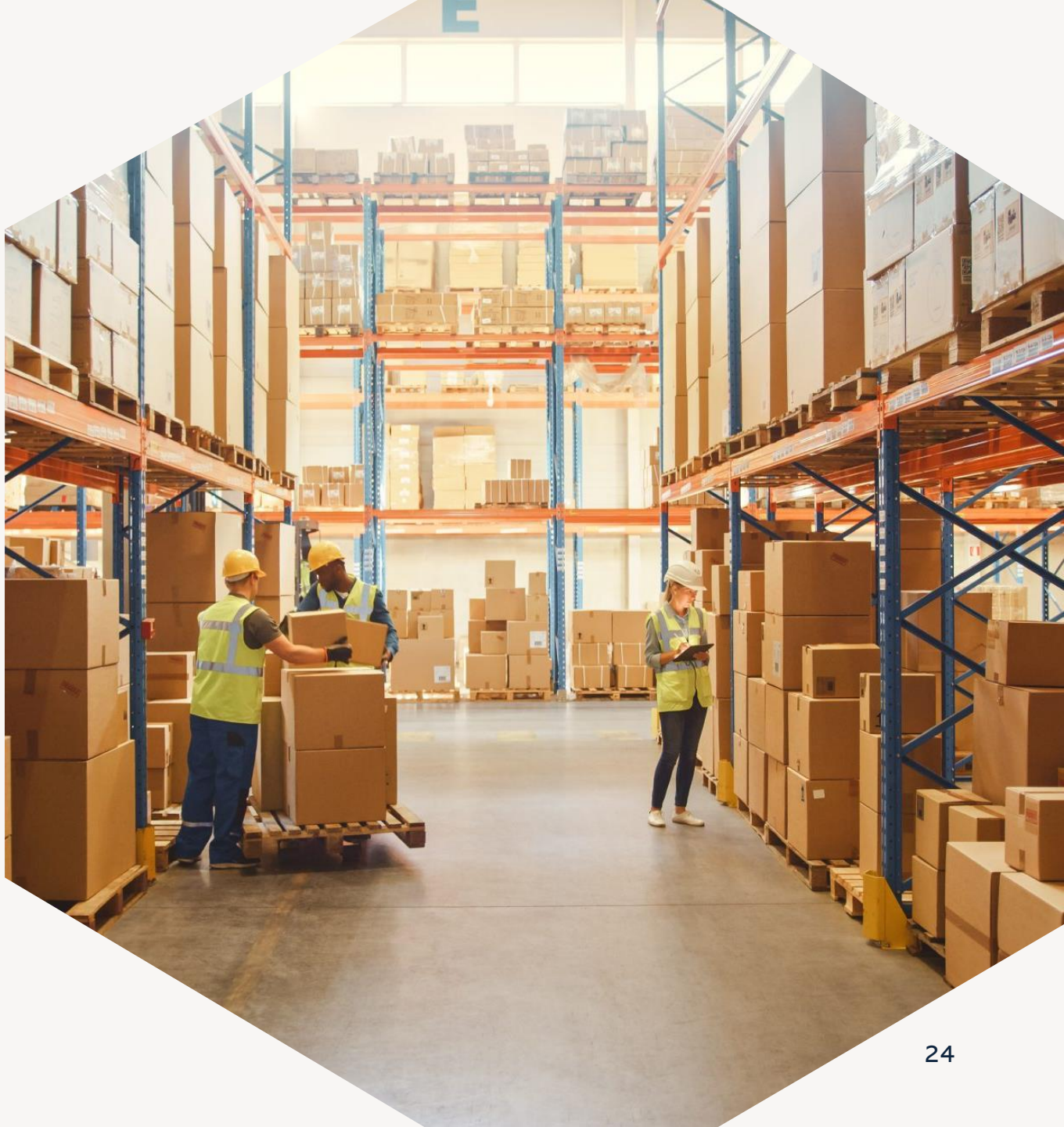


Aspect 02

Order picking route optimization

Introduction

- When a Warehouse user has multiple items to get when fulfilling an order, it can be a time-consuming process, with efficient routes, we can reduce time and energy wasted by the worker.
- This main goal is to implement a pathfinding algorithm used to find the most efficient route for picking orders in a warehouse, considering time, effectiveness, flagged obstacles and other workers and vehicles operating on-site.



Research Problem

- How can we optimize worker productivity and efficiency by mapping the path the user needs to take on the Warehouse

Objectives

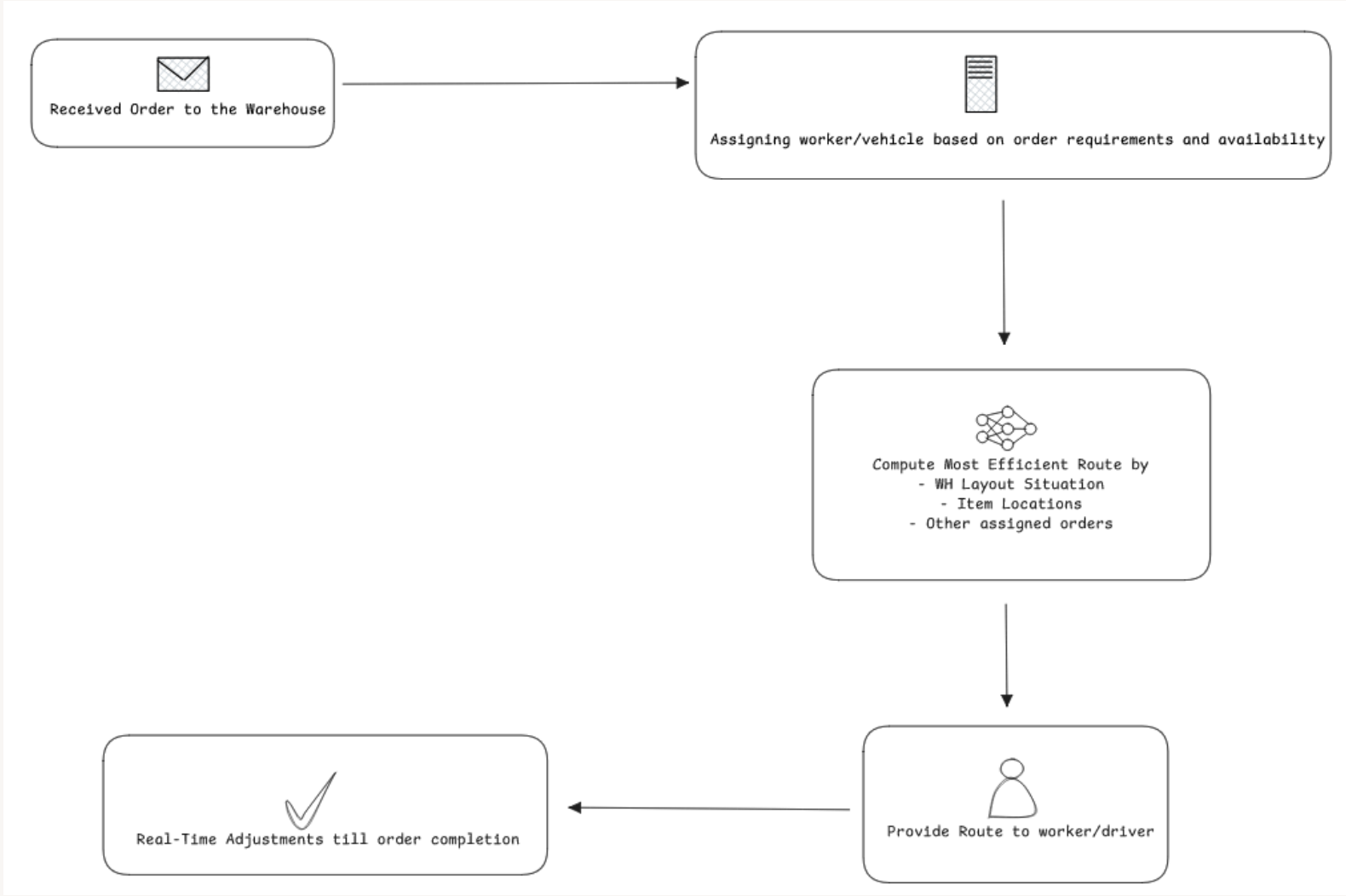
Main Objective

- Develop an efficient and real-time route optimization system for warehouse order pickers and forklifts, using an Algorithm and GPS tracking, to reduce time, avoid collisions, and improve operational efficiency.

Sub Objectives

- Implement the Algorithm which will compute the shortest most efficient path for the user.
- Using tracking tech to record the positions of workers and forklifts in real time, adjusting paths dynamically as needed.
- Map the routes of the warehouse, to the warehouse layout.

Methodology



Progress as Now

- **Order Picking Optimal Path**
  - Compute the shortest & efficient path to get from starting point to picking locations and back to the starting point.

- **Tech Stack :**
  - Python
  - Matplotlib (For animation and drawing objects)
  - OR-Tools (Aiding to solve the TSP)

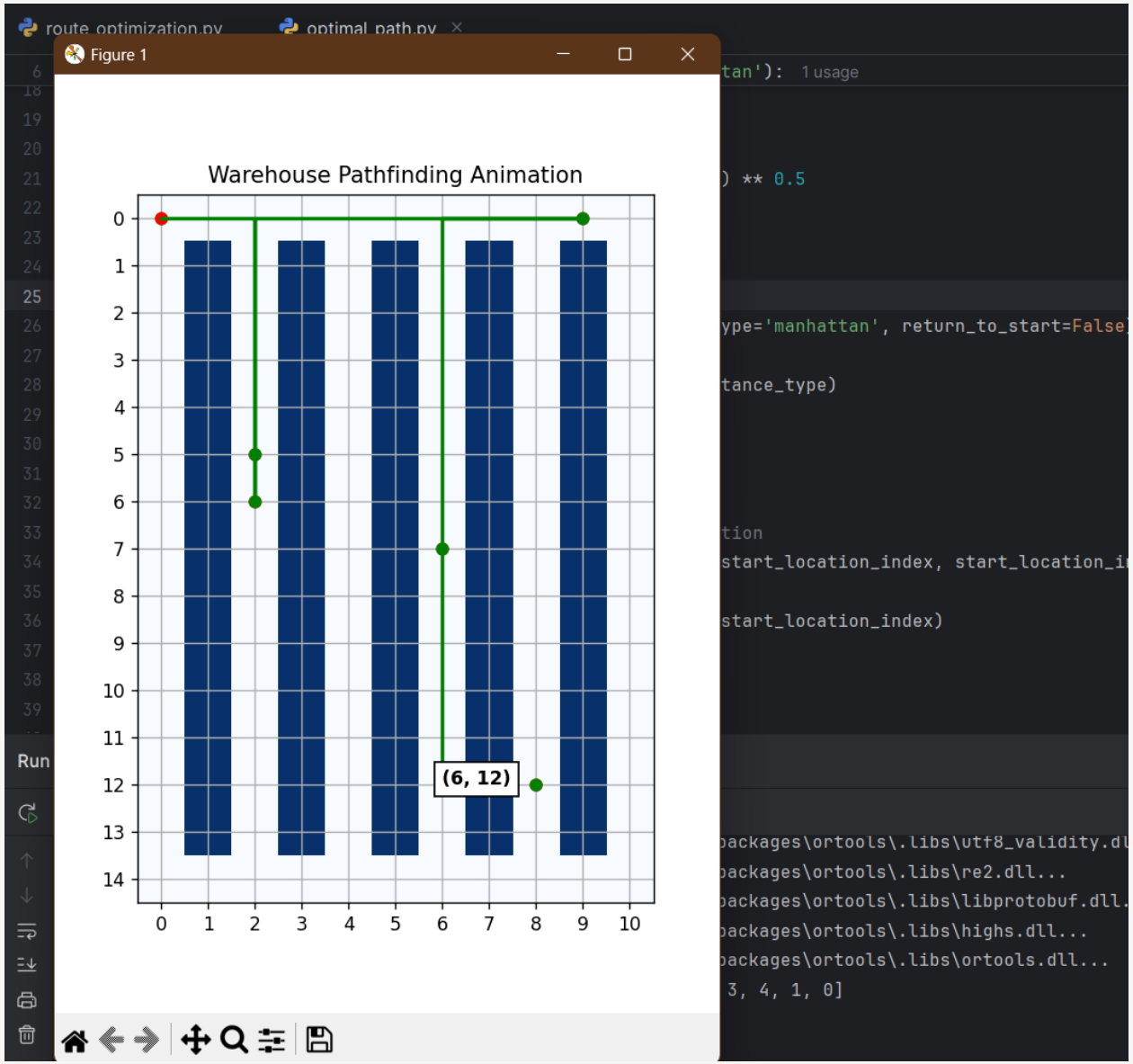
Travelling Salesman Problem (TSP)

```python
def solve_tsp(locations, start_location_index=0, distance_type='manhattan', return_to_start=False):  1 usage
    # Build distance matrix
    distance_matrix = create_distance_matrix(locations, distance_type)
    n = len(locations)

    # Create the routing index manager
    if return_to_start:
        # Force the route to start and end at the same location
        manager = pywrapcp.RoutingIndexManager( *args: n, 1, start_location_index, start_location_index)
    else:
        manager = pywrapcp.RoutingIndexManager( *args: n, 1, start_location_index)

    # Create the Routing Model
    routing = pywrapcp.RoutingModel(manager)

    def distance_callback(from_index, to_index):
        # Convert from routing variable Index to our location index
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return int(distance_matrix[from_node][to_node])

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Set parameters
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()

    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
    )

    # Solve the TSP travel salesmen problem
    solution = routing.SolveWithParameters(search_parameters)
```

Customized A* Algorithm

```python
def a_star(warehouse, start, goal):  2 usages
    open_list = []
    closed_list = set()
    g_scores = {start: 0}
    f_scores = {start: np.linalg.norm(np.array(start) - np.array(goal))}
    came_from = {}

    heapq.heappush( *args: open_list, (f_scores[start], start))

    while open_list:
        _, current = heapq.heappop(open_list)

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            path.reverse()
            return path

        closed_list.add(current)

        # Check the neighbors (up, down, left, right) allowed only orthogonally moves
        neighbors = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        for dx, dy in neighbors:
            neighbor = (current[0] + dx, current[1] + dy)
            if 0 <= neighbor[0] < warehouse.shape[0] and 0 <= neighbor[1] < warehouse.shape[1]:
                if warehouse[neighbor[0], neighbor[1]] == 1:  # Skip shelves (blocked paths)
                    continue
                if neighbor in closed_list:
                    continue

                tentative_g_score = g_scores.get(current, float('inf')) + 1
```

Output


route_optimization.py
optimal_path.py
Figure 1
Warehouse Pathfinding Animation
(6, 12)
Run

Technologies

Next.Js

AWS Lambda

Flask

Google OR

Python

PostgreSQL

Research Gap

- **Path Mapping**
  - Mapping the path to fulfill the order, is a function that is not available in the existing system
- **Dynamic Path Adjustment:**
  - Current systems do not dynamically adjust to real-time changes in the warehouse environment, such as congestion or maintenance zones.
- **Collision Avoidance:**
  - There is limited research on real-time collision avoidance between order pickers and forklifts in large warehouse environments.
- **Integration of GPS and Pathfinding:**
  - Few systems integrate **real-time GPS tracking** with **optimized pathfinding algorithms** like Dijkstra's to provide a seamless, real-time solution.

Novelty

Novelty of this system lies in its dynamic, real-time optimization of warehouse navigation. Key novel aspects include:

- Custom Path Finding Algorithm
- Real-Time Path Adjustments
- Collision Avoidance
- Maintenance Zone Integration

References

- Li, X., et al. (2023). "Optimized Pathfinding in Smart Warehouses." *International Journal of Logistics Management*.
- Y. Zhang, L. Zhang, and W. Li, "Real-time Dynamic Path Planning in Automated Warehouses," *Journal of Warehouse Technology*, 2021.
- "GPS Tracking in Warehouse Management: Real-time Positioning and Path Optimization," *Journal of Supply Chain Management*, 2022.

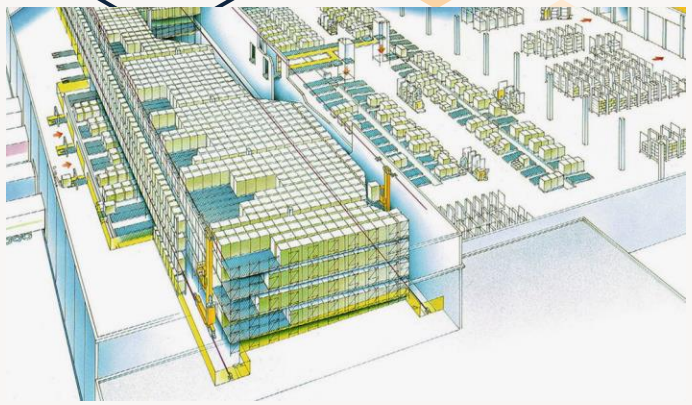
Aspect 03

Fire Detection Real-Time Alerts using computer vision

Introduction

- **Current Fire Detection Limitations**
- Traditional fire detectors, like smoke alarms, can't tell where the fire is, especially in areas with high risk, such as near racks or flammable materials.
- **Proposed AI Solution**
- want to create a fire detection system that uses AI and computer vision. This system can spot fires and assess how serious they are, especially near important areas .
- **Impact and Benefits**
- This system will improve fire safety by providing real-time alerts, faster responses, and more accurate risk assessments. It will help reduce damage and make things safer overall.

Research Problem



•Limited Fire Severity Detection

•Traditional fire systems can't determine how serious a fire is, especially if it's close to important areas like racks.

•Challenges in Complex Layouts

•In places like warehouses and factories, traditional systems often have trouble detecting fires accurately due to complex building layouts.

•Lack of AI Solutions

•AI and computer vision are not commonly used for real-time fire detection, especially in industrial settings in Sri Lanka.

Objectives

Main Objective

- **Develop a real-time fire detection and alert system using computer vision and AI** to detect fires, assess their severity based on proximity to high-risk zones, and provide real-time alerts for quicker responses and minimized damage.

Sub Objectives

- **Fire Detection**: Develop a deep learning model to detect fires in real-time from video feeds.
- **Fire Severity Classification**: Create a system to assess fire severity based on its proximity to critical areas like racks or equipment.
- **Real-time Alerts**: Build an alert system that notifies safety personnel about the fire's location, severity, and required actions.
- **Testing and Optimization**: Test the system in real-world environments to ensure accuracy, minimize latency, and ensure efficient operation for large-scale use.

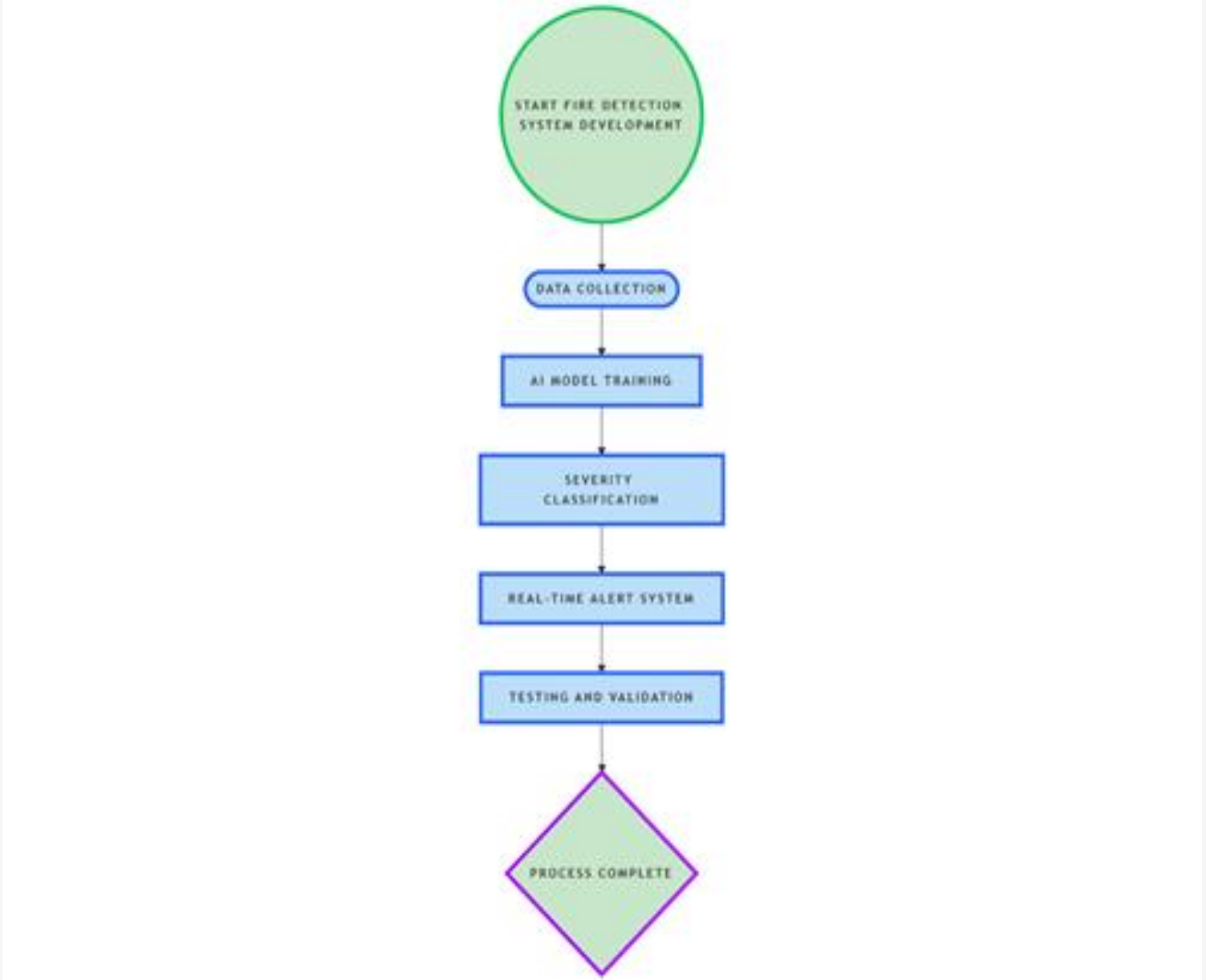
Research Gap

Aspect	Current system	Proposed AI-Based System
Detection Method	Smoke detectors, alarms	AI-powered computer vision(camera) for real-time fire detection
Impact Evaluation	No severity classification based on location	severity classification based on proximity to racks and critical areas
Real-time Alerts	Traditional alarms with manual response	Automated real-time alerts with detailed severity information
Technology Used	Smoke and heat detectors	AI, ,computer vision
Complex Layout Adaptability	Limited detection in complex, obstructed environments	AI detects fires in complex environments with obstacles

Novelty

This research presents a new fire detection system that uses AI and computer vision to spot fires in real-time and assess their danger based on their proximity to critical areas, like racks. Unlike traditional systems that only detect smoke, this system sends immediate alerts with detailed information about the fire's location and severity. **This makes it more effective in complex environments, warehouses. This technology is especially important in Sri Lanka, where it is not used.**

Methodology



Functional and Non-Functional Requirements

Functional Requirements:

- Fire Detection
- Fire Impact Classification
- Alert System
- Real-Time Processing

Non-Functional Requirements:

- Scalability
- Efficiency
- Reliability
- User Interface

What I did so far

Real-Time Object Detection & Depth Estimation:

- **Function**: Detects shelves and fire in images, estimates their depth, and alerts when objects are too close.

Tech Stack:

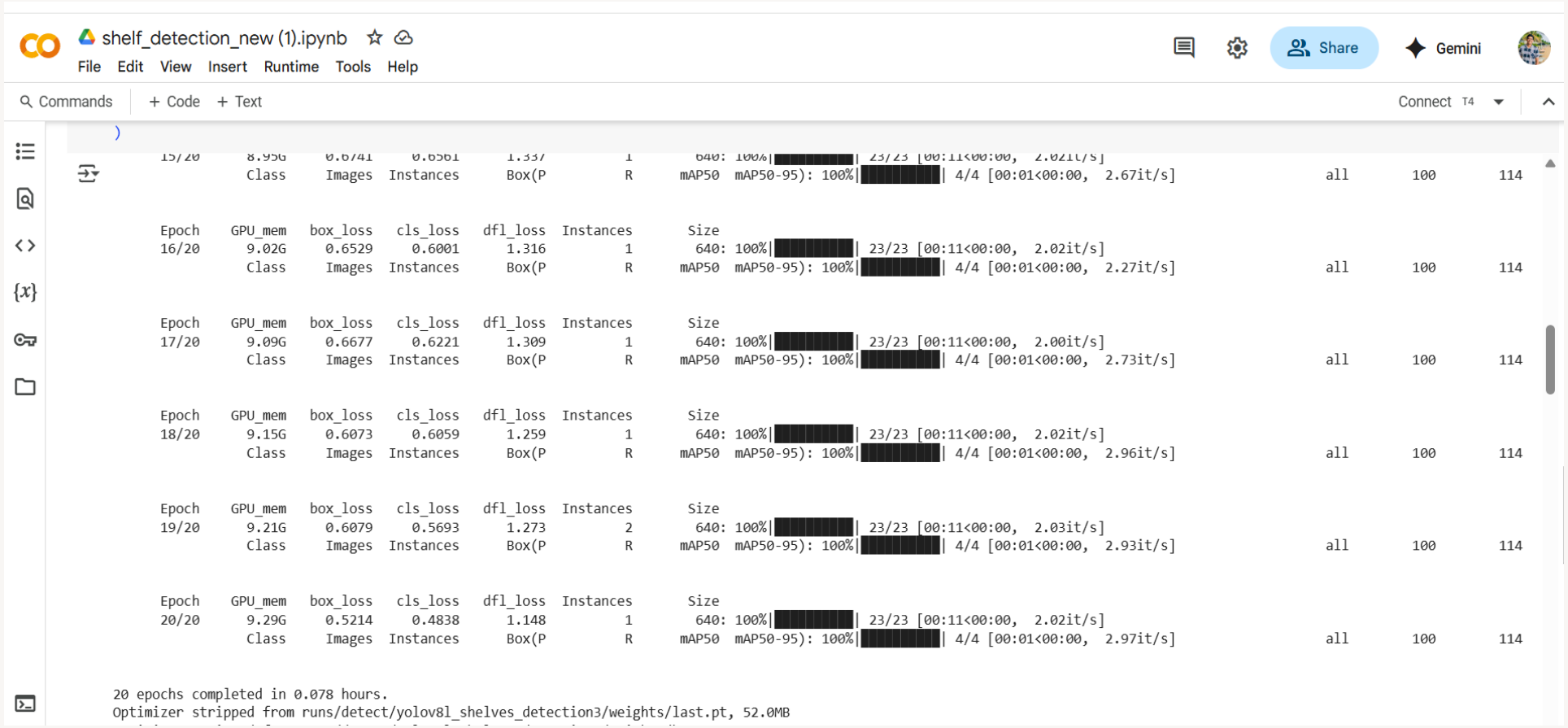
- **Python**
- **PyTorch** (for MiDaS and YOLO models)
- **OpenCV** (for image processing and visualization)

Code snippet of data set

```
[ ]  from roboflow import Roboflow
     rf = Roboflow(api_key="mQkB7HoQJGcIPhFY72LA")
     project = rf.workspace("fire-detection").project("fire-detection-data-pre")
     version = project.version(4)
     dataset = version.download("yolov8")
```

I used yolo8 for balanced real-time and accuracy

YOLOv8 Training Log for Shelf Detection



shelf_detection_new (1).ipynb

File Edit View Insert Runtime Tools Help

Share Gemini

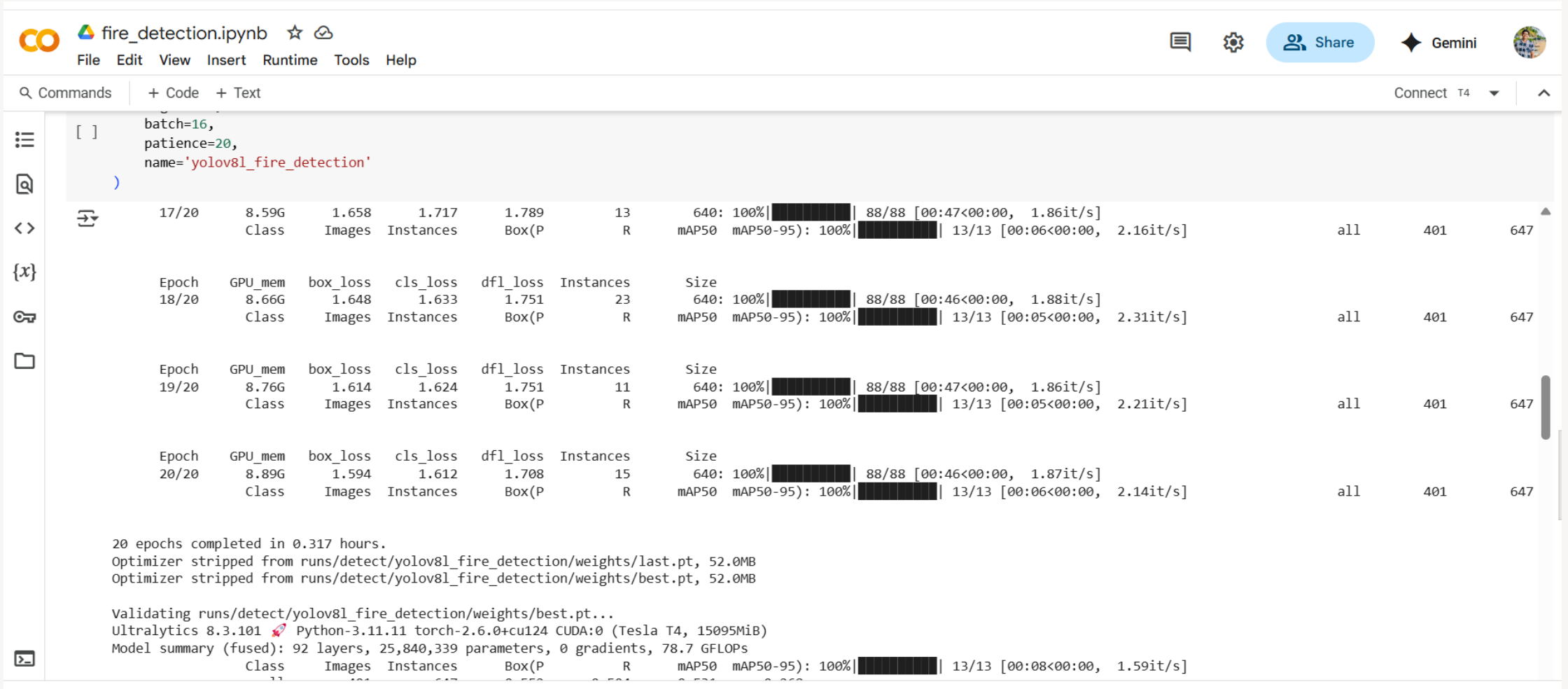
Commands + Code + Text Connect T4

```
)
```

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.67it/s]   all        100        114

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
16/20      9.02G     0.6529     0.6001      1.316          1        640: 100%|██████████| 23/23 [00:11<00:00,  2.02it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.27it/s]   all        100        114

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
17/20      9.09G     0.6677     0.6221      1.309          1        640: 100%|██████████| 23/23 [00:11<00:00,  2.00it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.73it/s]   all        100        114

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
18/20      9.15G     0.6073     0.6059      1.259          1        640: 100%|██████████| 23/23 [00:11<00:00,  2.02it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.96it/s]   all        100        114

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
19/20      9.21G     0.6079     0.5693      1.273          2        640: 100%|██████████| 23/23 [00:11<00:00,  2.03it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.93it/s]   all        100        114

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
20/20      9.29G     0.5214     0.4838      1.148          1        640: 100%|██████████| 23/23 [00:11<00:00,  2.02it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.97it/s]   all        100        114

20 epochs completed in 0.078 hours.
Optimizer stripped from runs/detect/yolov8l_shelves_detection3/weights/last.pt, 52.0MB
```

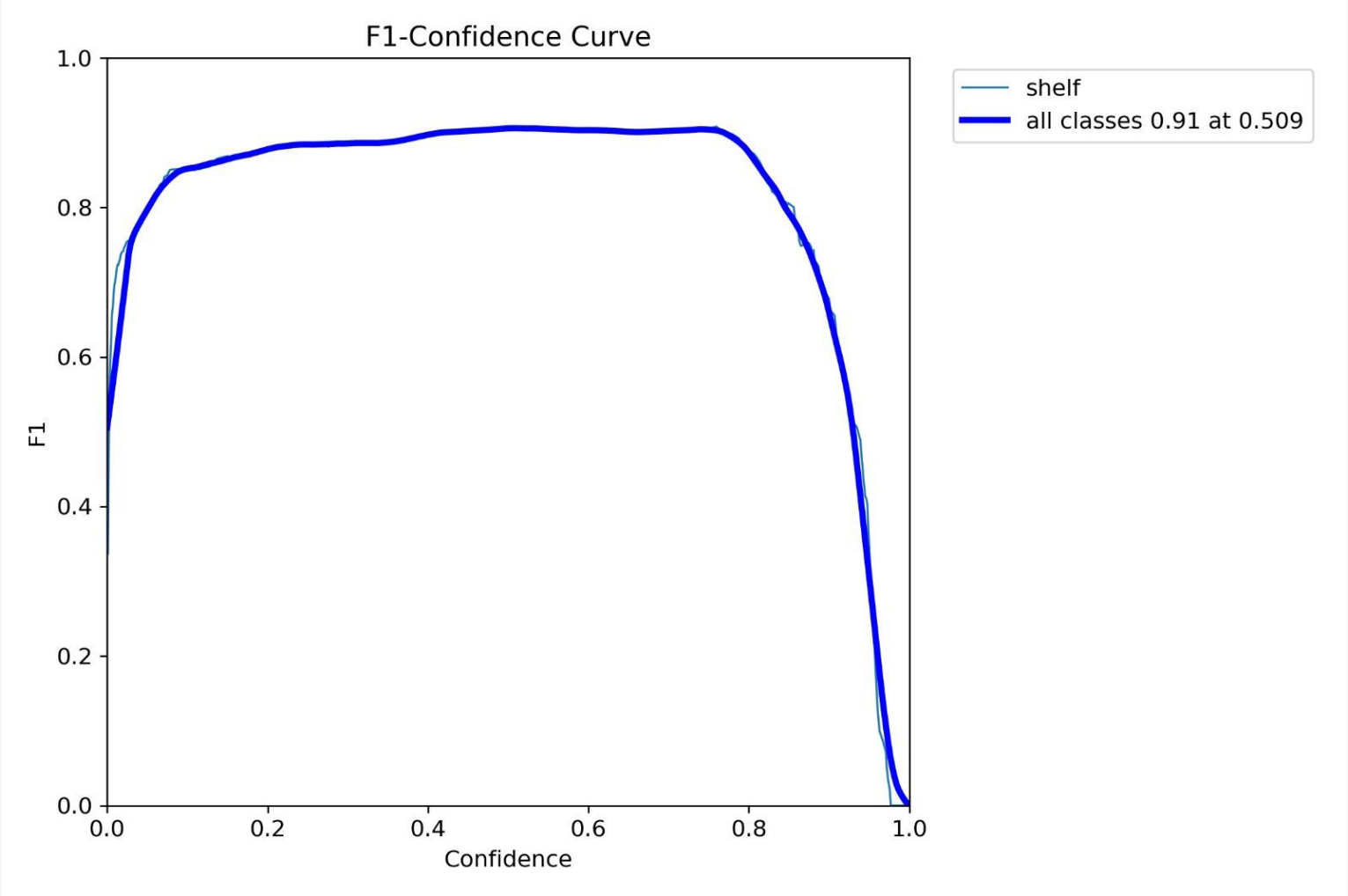
The model has completed 20 epochs. Trained on **Google Colab**

YOLOv8 Fire Detection Model Training

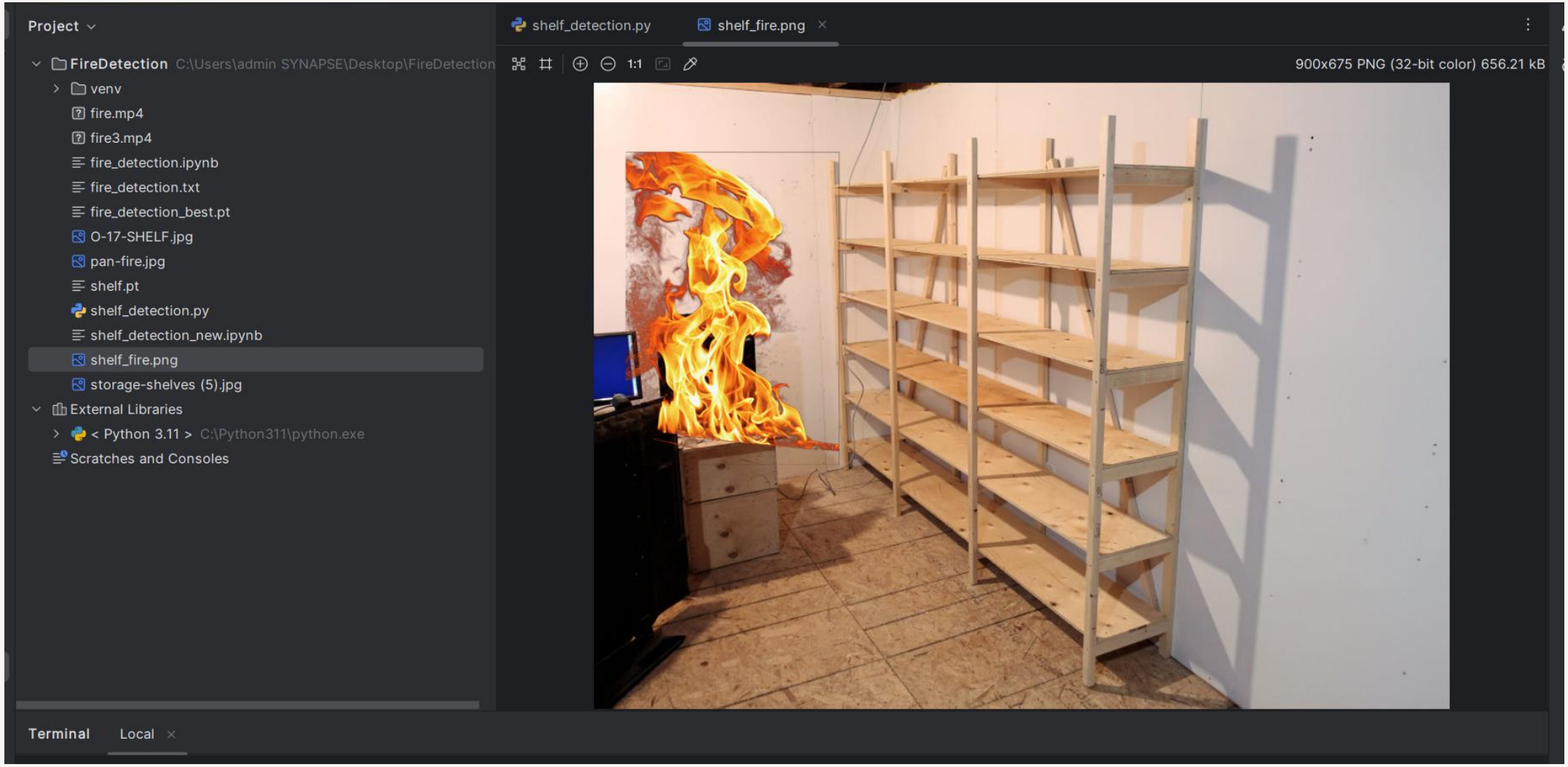


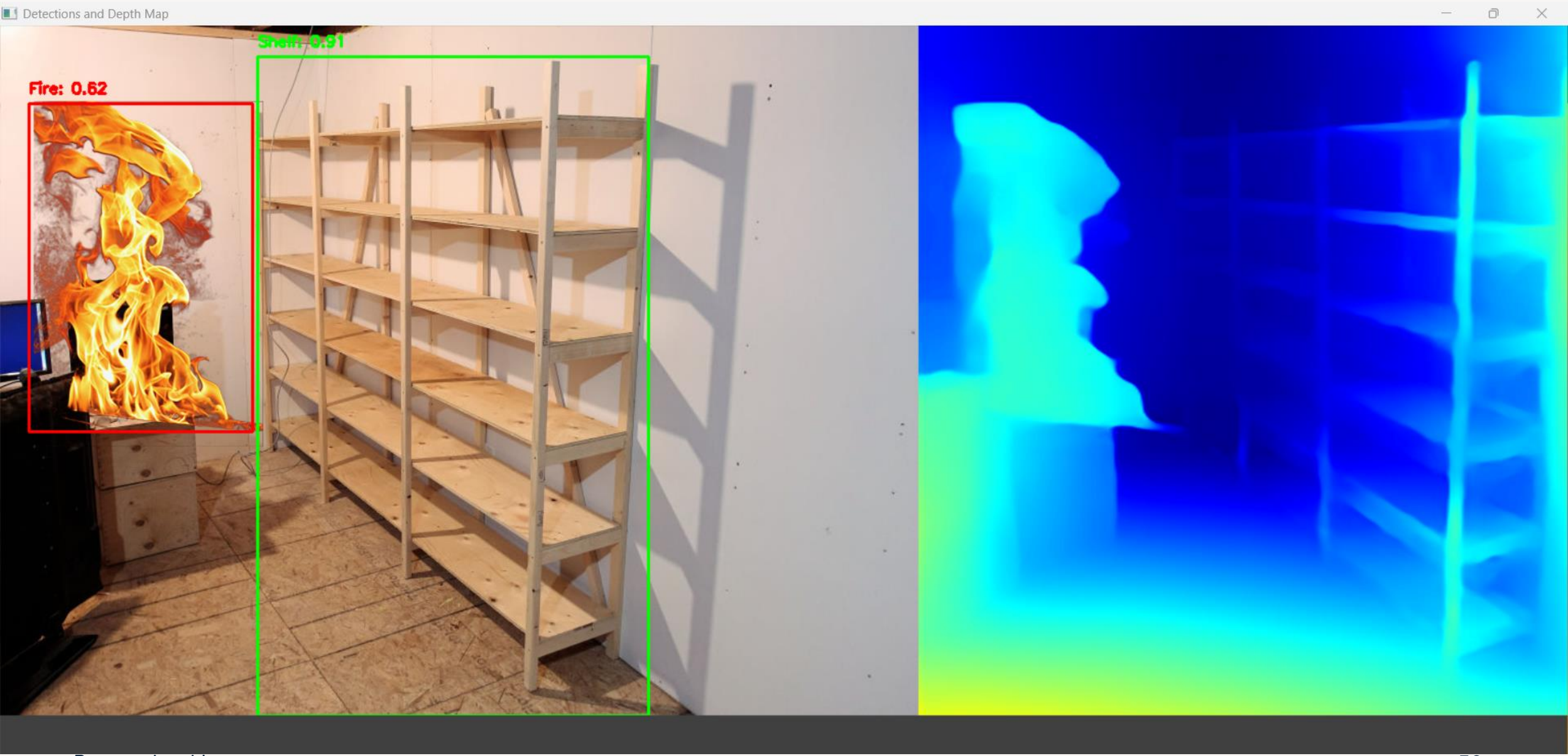
The model has completed 20 epochs. Trained on **Google Colab**

Model Accuracy



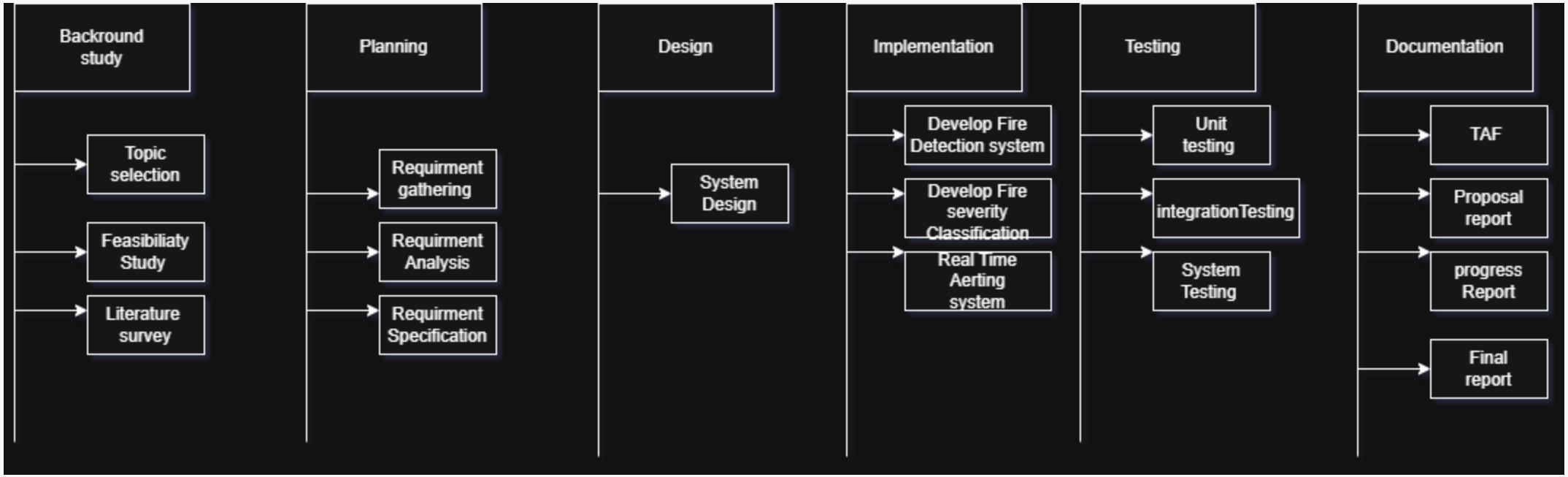
output


Project
FireDetection C:\Users\admin SYNAPSE\Desktop\FireDetection
venv
fire.mp4
fire3.mp4
fire_detection.ipynb
fire_detection.txt
fire_detection_best.pt
O-17-SHELF.jpg
pan-fire.jpg
shelf.pt
shelf_detection.py
shelf_detection_new.ipynb
shelf_fire.png
storage-shelves (5).jpg
External Libraries
< Python 3.11 > C:\Python311\python.exe
Scratches and Consoles
shelf_detection.py
shelf_fire.png
1:1
900x675 PNG (32-bit color) 656.21 kB
Terminal
Local


Detections and Depth Map
Shelf: 0.91
Fire: 0.62

Tools and technology

Work Break-Down Structure



References

Fernando, M. (2020). *Fire Safety Systems in Sri Lankan Industries: Challenges and Opportunities*. Journal of Industrial Safety, 12(3), 45-58.

- Wickramasinghe, S., & Gunawardena, T. (2021). *Computer Vision for Industrial Safety: Applications in Sri Lanka*. Proceedings of the International Conference on AI and Automation.
- Kumar, R., & Patel, A. (2019). *Fire Detection and Surveillance Systems: A Review*. Journal of Artificial Intelligence, 31(7), 1021-1037.
- Zeng, H., & Wang, D. (2018). *Real-Time Fire Detection Using Deep Learning*. International Journal of Computer Vision and Pattern Recognition, 20(5), 101-114.
- Mohan, S., & Das, P. (2022). *The Role of AI in Enhancing Fire Safety: A Review*. International Journal of AI Applications, 16(2), 65-82.

Aspect 04

stock movement instabilities by analyzing historical inventory data.

Introduction

- This system provides real-time alerts for sudden stock movement instabilities by analyzing historical inventory data. When a movement exceeds the normal range (based on average past behavior), the system triggers an real-time alert and visually highlights the affected rack location on a warehouse map. This allows warehouse staff to immediately investigate and resolve potential issues, enhancing both efficiency and accuracy in inventory management.

Research Problems

Threshold Determination:

•There is limited research on effectively setting statistical thresholds based on historical data to distinguish normal from abnormal stock movement.

Integration with Historical Data:

•Current inventory systems often focus on current data, leaving a gap in predictive analysis using past data to drive proactive decision-making.

Alert Accuracy:

•Without careful analysis, systems may trigger too many false alarms, How can statistical or machine learning techniques be applied to minimize false positives and negatives when triggering alerts for inventory movements

Main Objective

●To develop a predictive inventory management system that leverages historical inventory movement data to detect anomalous stock movements and give real-time alerts.

Sub Objectives

Data Aggregation and Preprocessing:

- Collect and integrate historical inventory movement records from existing WMS systems.
- Develop data cleaning and normalization processes to ensure high-quality inputs for analysis.

Baseline Modeling and Anomaly Detection:

- Establish dynamic statistical baselines (e.g., using time-series forecasting) for normal stock movements.
- Implement and optimize anomaly detection algorithms to flag movements that exceed the baseline thresholds.

Visualization and Alerting Mechanism:

- Design a user-friendly dashboard that visually maps anomalies onto warehouse rack layouts.
- Integrate automated notification systems (e.g., email, SMS) to alert managers when anomalies are detected.

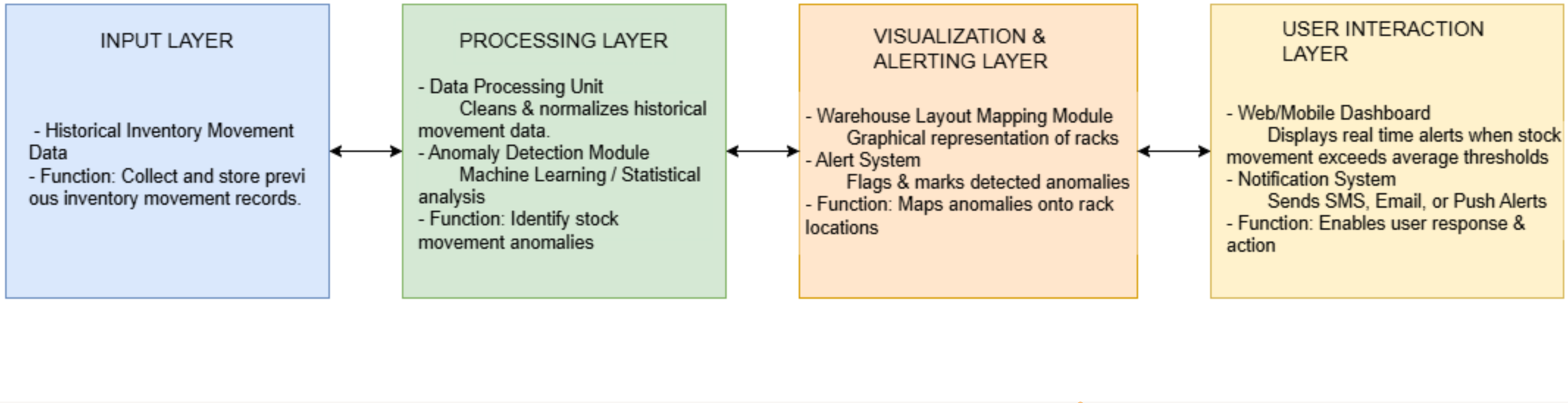
System Evaluation and Performance Optimization:

- Evaluate the system's prediction accuracy and responsiveness using defined metrics (e.g., false positive/negative rates, cost savings, and response times).

Research Gaps

- **Leverage Historical Data:** Few studies have investigated the use of historical inventory movement data for predictive anomaly detection.
- **Threshold Determination:** Limited research exists on establishing statistically sound thresholds to trigger alerts based on past data.
- **Cost-Effective Solutions:** There is an opportunity to develop systems that provide actionable insights without the cost and complexity of installing and maintaining real-time IoT sensor networks.

Methodology



Functional Requirements

- **Historical Data Collection:**
- Ability to gather historical inventory movement data from ERP/WMS systems.
- **Data Preprocessing:**
- Data cleaning, normalization, and integration processes.
- **Forecasting and Baseline Modeling:**
- Load pre-trained forecasting models for both inbound and outbound quantities and generate future forecasts based on historical trends and apply rolling averages to smooth predictions.
- **Deviation/Anomaly Detection:**
- Compare real observed data against forecasted values and flag deviations where actual dispatch or inbound quantities differ from forecasted values by more than a configurable percentage threshold.
- **Visualization and Mapping:**
- Display a digital warehouse layout with rack-level mapping of anomalies.
- Graphical representations (charts, heat maps) of historical and predicted inventory trends.
- **Alerting and Notification:**
- Automated alert system that triggers notifications (email, SMS, in-app) when anomalies are detected.
- Options for users to customize alert thresholds and frequency.

Non-Functional Requirements

- **Performance:** The system should process data and generate alerts within a predefined response time (e.g., under 5 seconds for each anomaly detection cycle).
- **Scalability:** Must handle large volumes of historical data and scale with growing data size and number of inventory nodes.
- **Reliability and Availability:** Ensure continuous system operation with high uptime (e.g., 99.9% availability).
- **Security:** Ensure data confidentiality and integrity with encryption and access controls.
- **Usability:** User-friendly dashboard with intuitive navigation and clear visualizations. And minimal training required for warehouse staff to effectively use the system.

Current Progress

- **Data Preparation & Aggregation:**
- Filtered and cleaned historical inventory data based on product category and specific warehouse (e.g., "Warehouse 1").
- **Model Training for Each Category:**
- Initialized two forecasting models (using Prophet) per category - one for outbound data and one for inbound data with parameters optimized for smoother trend detection and to capture seasonal effects.
- **Model Serialization:**
- Saved the trained models to persistent storage so that they can be later loaded and used for forecasting.
- **Forecast Model Integration:**
- Developed a Python function that forecasts both inbound and outbound inventory quantities using pre-trained forecasting models (Prophet) specific to each product category.

- **Data Preparation & Forecasting:**
- Generated future date ranges (weekly) and applied the Prophet models to predict inventory movement.
- Used a rolling average technique to smooth the forecasted data, improving trend detection and making the predictions more reliable.
- **Historical Data Comparison:**
- Filtered and aggregated historical inventory data by date and merged it with forecasted data for direct comparison.
- **Anomaly Detection:**
- Implemented a threshold-based mechanism that flags deviations of over 10% between actual and forecasted inventory levels.
- **Visualization & Reporting:**
- Created visual charts and summary tables that clearly display forecast trends, actual data, and marked anomalies for review.

Code snippets

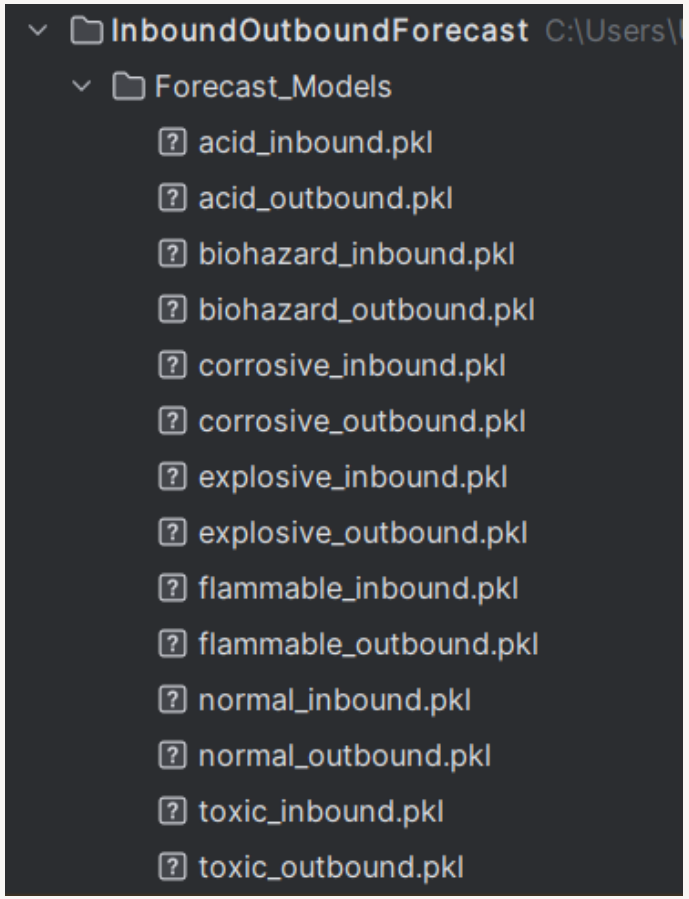
```python
for value in unique_values:

    selected_category = value
    df_category = df_realistic[(df_realistic["Category"] == selected_category) &
      (df_realistic['Warehouse'] == "Warehouse 1")].copy()

    df_category["Dispatch_Date"] = pd.to_datetime(df_category["Dispatch_Date"])
    df_time_series = df_category.groupby(pd.Grouper(key="Dispatch_Date", freq="W"))[["Dispatch_Quantity", "Inbound_Quantity"]].sum().reset_index()

    df_time_series["Dispatch_Date"] = pd.to_datetime(df_time_series["Dispatch_Date"])

    df_outbound = df_time_series.rename(columns={"Dispatch_Date": "ds", "Dispatch_Quantity": "y"})

    # Apply log transformation to stabilize variance
    df_outbound["y"] = np.log1p(df_outbound["y"])

    # Initialize and train the Prophet model with smoother trend detection
    model_outbound = Prophet(changepoint_prior_scale=0.03, yearly_seasonality=True)
    model_outbound.fit(df_outbound)

    # Prepare data for Prophet
    df_inbound = df_time_series.rename(columns={"Dispatch_Date": "ds", "Inbound_Quantity": "y"})

    # Apply log transformation to stabilize variance
    df_inbound["y"] = np.log1p(df_inbound["y"])

    # Initialize and train the Prophet model for inbound forecasting
    model_inbound = Prophet(changepoint_prior_scale=0.03, yearly_seasonality=True)
    model_inbound.fit(df_inbound)

    # saving the models
    # Save Outbound Forecast Model
    joblib.dump(model_outbound, f"{selected_category}_outbound.pkl")
    # Save Inbound Forecast Model
    joblib.dump(model_inbound, f"{selected_category}_inbound.pkl")
```

Data cleaning and save the models

```python
import joblib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def predict_forecast_for_a_category(category, df_real_data, start_month, end_month, start_week=1, end_week=5, year=pd.Timestamp.now().year, time_frame=60, threshold=0.1):
    """
        category: The category model to forecast.
        df_real_data: The actual historical data containing 'Dispatch_Date', 'Dispatch_Quantity', and 'Inbound_Quantity'.
        start_month: The starting month (1-12).
        end_month: The ending month (1-12).
        start_week: The starting week of the month (1-5). Defaults to 1 optional.
        end_week: The ending week of the month (1-5). Defaults to 5 optional.
        year: The year for prediction. Defaults to the current year.
        time_frame: Number of weeks to predict. Defaults to 60.
        threshold: Percentage threshold for deviation detection. Default is 0.1 (10%).
    """
    directory = "Forecast_Models"

    # Load saved Prophet models
    model_outbound_loaded = joblib.load(f"{directory}/{category}_outbound.pkl")
    model_inbound_loaded = joblib.load(f"{directory}/{category}_inbound.pkl")

    # Generate future dates for forecasting (weekly)
    future_dates = model_outbound_loaded.make_future_dataframe(periods=time_frame, freq="W")

    # Make predictions using the loaded models
    forecast_outbound = model_outbound_loaded.predict(future_dates)
    forecast_outbound["yhat"] = np.expm1(forecast_outbound["yhat"])  # Reverse log transformation

```

Predict_forecast_for_a_category function for forecast predictions

```python
if __name__ == "__main__":
    df_realistic = pd.read_csv("WareHouseDataset.csv")
    predict_forecast_for_a_category( category: "flammable",df_realistic,start_month=3,end_month=7,time_frame=5,year=2024)
```



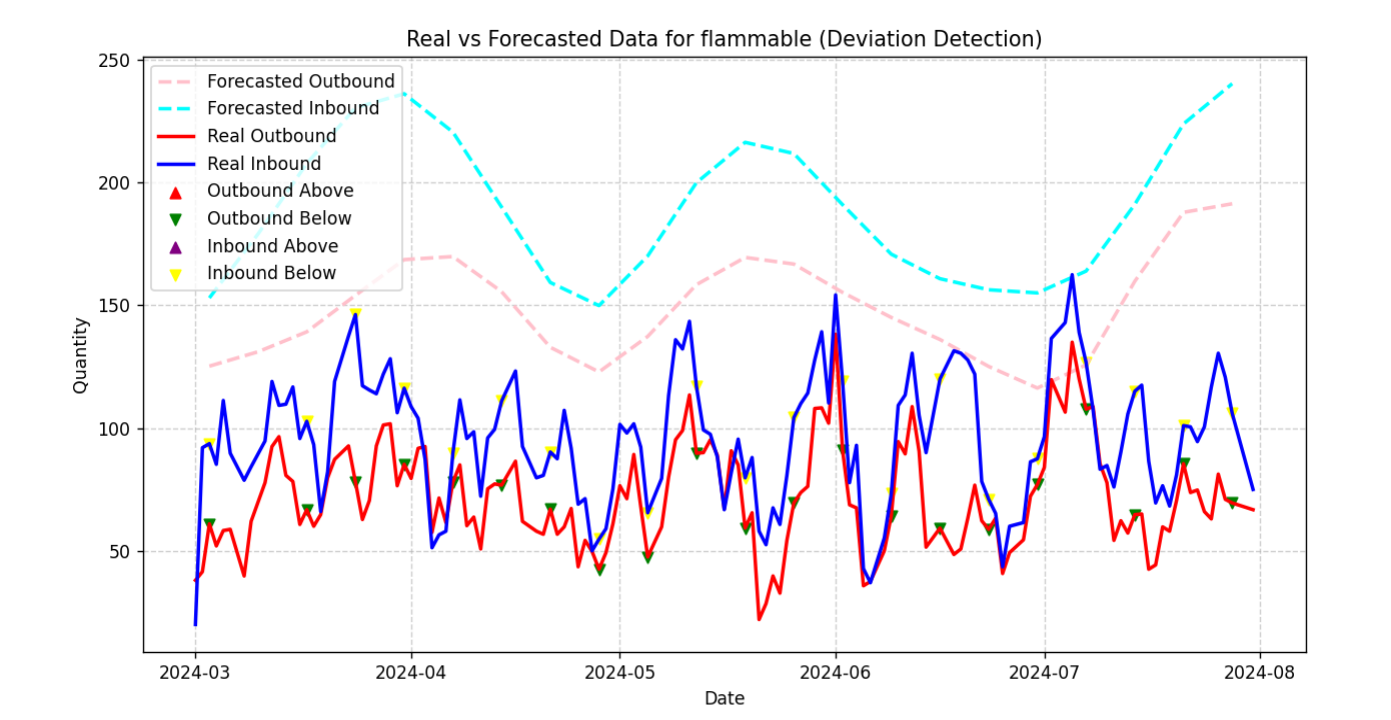
Forecast models

Model Accuracy

```
Inbound model for fla
mae      24.817428
rmse     33.579580
mape      3.813595
dtype: float64
```

Model has a 3.81% error rate and 96.19% accuracy

Proof of Results



Comparison of Real vs Forecasted Inbound & Outbound Quantities for flammable:

	ds	Inbound_Forecast	...	Dispatch_Quantity	Inbound_Quantity
0	2024-03-03	153.040162	...	61.00	93.666667
1	2024-03-10	179.106373	...	NaN	NaN
2	2024-03-17	207.798344	...	66.75	102.750000
3	2024-03-24	230.533900	...	78.00	146.250000
4	2024-03-31	236.222517	...	85.25	116.250000
5	2024-04-07	220.619720	...	78.00	89.750000
6	2024-04-14	190.050054	...	76.75	111.000000
7	2024-04-21	159.319753	...	67.00	90.250000
8	2024-04-28	149.913523	...	42.50	54.750000
9	2024-05-05	170.147660	...	47.50	65.500000
10	2024-05-12	199.988923	...	89.75	117.000000
11	2024-05-19	216.423871	...	59.25	79.750000
12	2024-05-26	211.878579	...	69.50	104.250000
13	2024-06-02	191.032928	...	91.00	119.000000
14	2024-06-09	170.862394	...	64.25	73.250000
15	2024-06-16	160.860898	...	59.25	120.000000
16	2024-06-23	156.353166	...	58.75	71.000000
17	2024-06-30	155.082222	...	77.00	87.500000
18	2024-07-07	163.970463	...	107.75	126.750000
19	2024-07-14	191.163301	...	64.75	115.000000
20	2024-07-21	223.943888	...	85.50	101.000000
21	2024-07-28	240.313760	...	69.50	106.000000

[22 rows x 5 columns]

Tools & Technologies

References

1. Fang, X., & Chen, H. C. (2022). *Using vendor management inventory system for goods inventory management in IoT manufacturing*. Enterprise Information Systems, 16(7).

2. Teerasoponpong, S., & Sopadang, A. (2022). *Decision support system for adaptive sourcing and inventory management in SMEs*. Robotics and Computer-Integrated Manufacturing.

3. Maheshwari, P., et al. (2021). *Internet of Things for Perishable Inventory Management Systems: Application and Managerial Insights for SMEs*. Annals of Operations Research, 1-29.

4. PackageX Blog. (2024). *Real-Time Inventory Management – Benefits and Challenges*.

5. Inoxoft. (2025). *7 Reasons to Invest in Real-Time Inventory System for Your Warehouse Operations*.

Budget

Hardware Costs:

- Cameras, Thermometers – LKR 30,000

Hosting Costs:

- AWS s3 instance – LKR 30,000 * 12
- AWS Lambda instances – LKR 20,000 * 12

Other Expenses

- Misc – LKR 10,000

Total Estimated Budget: LKR 90,000/=

Project Requirements

Functional Requirements

1. Real-Time Monitoring: The system must enable users to monitor warehouse operations, including product arrangements and space utilization, in real time.
2. Dynamic Product Storage Optimization: Use Machine Learning models to dynamically suggest optimal product arrangements based on CBM, product turnover rates, and constraints.
3. Interactive User Interface: Develop an intuitive web-based application using React and Python Django (or Node.js with Express.js) for seamless user interaction.
4. Inventory Security Features: Provide safeguards for sensitive and high-value goods by implementing risk management measures such as access controls and alert systems.
5. Customizable Storage Strategies: Allow users to input constraints (e.g., product fragility, weight, or stacking rules) to tailor the storage optimization process.

Project Requirements

Non-Functional Requirements

1. Performance: The system should provide real-time responses for 3D visualization and storage optimization, with minimal latency.
2. Usability: The user interface must be intuitive, enabling users with varying levels of technical expertise to interact with the system effectively.
3. Scalability: The system should support increasing warehouse sizes, product volumes, and additional features without compromising performance.
4. Compatibility: Ensure compatibility across major web browsers and devices, including desktops, laptops, and tablets.
5. Reliability: The system must operate reliably under various conditions, ensuring data accuracy and uninterrupted service.

References

[1] T. D. Rupasinghe and S. Dissanayake (2018) An integrated warehouse design and optimization modelling approach to enhance supply chain performance.

[2] **Chung, S. H., & Lee, H. (2017).** "Optimization of warehouse space utilization for container storage." *Journal of Manufacturing Science and Engineering, 139*(2), 021004.

[3] Li, X., et al. (2023). "Optimized Pathfinding in Smart Warehouses." International Journal of Logistics Management.

[4] Y. Zhang, L. Zhang, and W. Li, "Real-time Dynamic Path Planning in Automated Warehouses," Journal of Warehouse Technology, 2021.

[5] **Zhao, X., & Lee, D. (2016).** "Predictive analytics for monitoring stock movements in warehouses." *International Journal of Production Research, 54*(10), 3023-3035.

[6] **Yang, Y., & Zhang, X. (2020).** "AI-based fire detection systems in warehouse management: A review." *Computers, Materials & Continua, 64*(2), 1071-1089

Thank you...