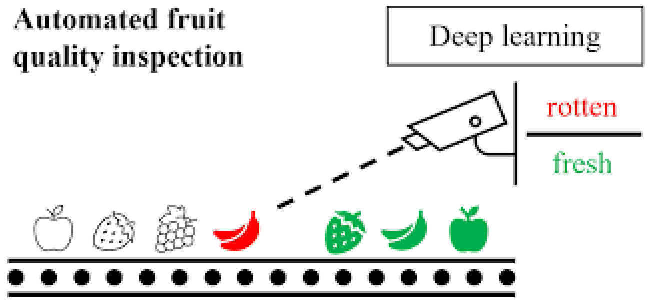
Project Design Phase-II
Technology Stack (Architecture & Stack)

Date	15 Jun 2025
Team ID	LTVIP2025TMID36983
Project Name	Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables
Maximum Marks	4 Marks

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2

Example: architecture of Small sorting: Transfer learning for identifying rotten in fruits and vegetables



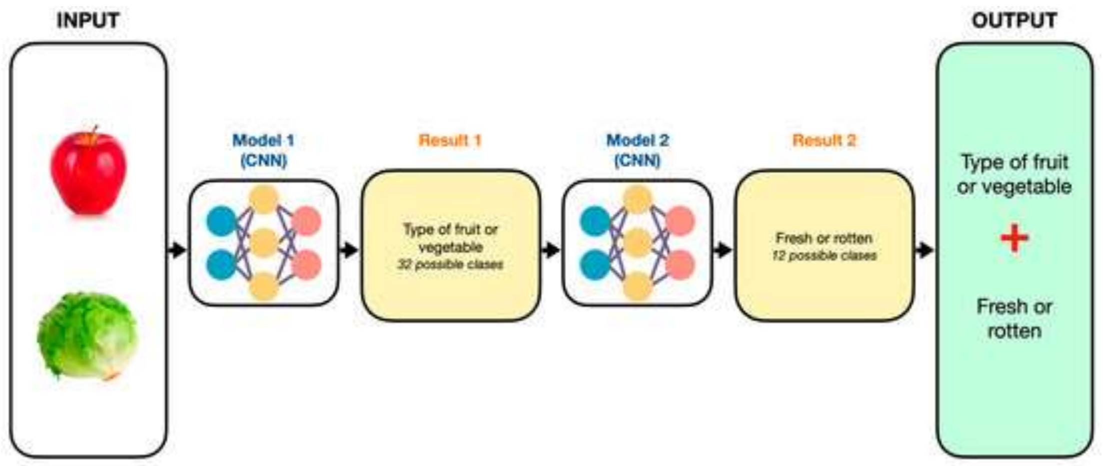


Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	Provides a simple way for users to monitor system status, view classification results (if applicable), and potentially configure basic settings or provide	**Local Display/LEDs:** Small LCD/OLED screen, indicator LEDs (e.g., green for fresh, red for rotten). **Mobile App/Web Interface:** (Optional,

		feedback. For small-scale, embedded systems, this might be a very minimal interface.	for advanced settings, remote monitoring, or feedback) HTML/CSS/JavaScript (React, Vue.js, Angular), Swift/Kotlin (for native mobile apps)
2.	Application Logic-1	Manages the camera hardware, captures still images or video frames of the fruits/vegetables, and handles initial image pre-processing (e.g., cropping, resizing to model input dimensions).	**Python:** OpenCV, Picamera (for Raspberry Pi), gPhoto2 (for DSLR/mirrorless cameras). **C++:** OpenCV.
3.	Application Logic-2	Implements the core logic for loading the trained machine learning model, feeding the pre-processed image to it, and obtaining the classification output (fresh/rotten) along with confidence scores.	**Python:** TensorFlow Lite, PyTorch Mobile, ONNX Runtime. **C++:** TensorFlow Lite C++ API, LibTorch.
4.	Application Logic-3	Interprets the classification result and activates the appropriate sorting mechanism (e.g., a servo motor, solenoid, or pneumatic actuator) to direct the fruit/vegetable to the "fresh" or "rotten" output bin. Includes logic for sequencing and timing.	**Python:** RPi.GPIO (for Raspberry Pi), smbus (for I2C sensors/actuators).
 C++: Arduino IDE (for microcontrollers), embedded C.
5.	Database	(Optional for very small-scale) Stores local configuration settings, basic operational logs, or a limited history of classification results for immediate access without network dependency.	**SQLite:** For lightweight, embedded local storage. **JSON/YAML files:** For simple configuration storage.
6.	Cloud Database	Stores aggregated classification data, user feedback, system performance metrics, and potentially updated model versions for centralized management and analysis. Enables remote monitoring and continuous improvement.	**Google Cloud Firestore/Datastore:** NoSQL document database. **AWS DynamoDB:** NoSQL key-value and document database. **MongoDB Atlas:** Man
7.	File Storage	Stores raw images (for retraining purposes), trained model files, application logs, and potentially firmware updates for deployment.	**Local Storage:** SD Card, eMMC, USB Flash Drive (for the embedded device). **Cloud Storage:** Google Cloud Storage, AWS S3, Azure Blob Storage (for large-scale data storage and access).
8.	External API-1	Used for user registration, login, and managing user permissions if the system includes a web or	**Google Firebase Authentication:** For easy integration with Gmail/social logins.

		mobile interface for advanced features or remote access.	**Auth0 / Okta:** Identity as a Service (IDaaS). **Custom REST API:** Built with Django REST Framework, Flask, Node.js (Express).
9.	External API-2	Facilitates the secure delivery of software and machine learning model updates to deployed sorting units, ensuring the latest features and improved accuracy.	**AWS IoT Core / Greengrass:** For secure device communication and edge deployment. **Google Cloud IoT Core:** For connecting and managing IoT devices. **Custom API/MQTT broker:** For specific update mechanisms.
10.	Machine Learning Model	The core AI component responsible for identifying rotten produce. It's a pre-trained deep learning model fine-tuned on a custom dataset.	**Transfer Learning Models:** MobileNetV2, ResNet, EfficientNet (chosen for efficiency and performance). **Frameworks:** TensorFlow Lite, PyTorch Mobile (for on-device inference).
11.	Infrastructure (Server / Cloud)	Provides the backend services for data storage, user management, model re-training (if performed in the cloud), and serving model updates to the devices. For the edge device itself, it's the computing platform.	**Edge Device:** Raspberry Pi, NVIDIA Jetson Nano, Coral Dev Board (for on-device inference). **Cloud Platform:** Google Cloud Platform (GCP), Amazon Web Services (AWS), Microsoft Azure (for backend services, data storage, and training).

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	The solution extensively leverages open-source machine learning frameworks and libraries to build, train, and deploy the classification model, reducing development costs and allowing for community support and flexibility.	**TensorFlow / Keras:** For building and training deep learning models. **PyTorch:** Another leading deep learning framework. **OpenCV:** For image processing and

S.No	Characteristics	Description	Technology
			computer vision tasks. **Python:** General-purpose programming language.
2.	Security Implementations	Security measures will be implemented to protect data (if transmitted to cloud), ensure the integrity of the deployed model, and prevent unauthorized access or tampering with the device. This is especially relevant if the device connects to a network.	**HTTPS/TLS:** For secure data transmission to cloud services. **API Keys / Tokens:** For authenticating cloud service requests. **Secure Boot / Signed Firmware:** (For embedded systems) To prevent unauthorized code execution. **Containerization (e.g., Docker/Podman):** To isolate application components.
3.	Scalable Architecture	The architecture is designed to accommodate increasing data volumes, more diverse produce types, and higher throughput demands. This includes both the model's ability to be retrained with new data and the system's ability to be replicated or integrated into larger sorting lines.	**Transfer Learning:** Allows quick adaptation to new produce types without training from scratch. **Modular Design:** Separation of concerns (image capture, inference, sorting control) enables independent scaling. **Cloud Infrastructure:** For scalable data storage, model training, and remote management. **Edge Computing:** Distributes processing to reduce latency and bandwidth usage, scaling out by adding more devices.
4.	Availability	The system is designed for high uptime, ensuring continuous operation for efficient sorting. Measures include robust error handling, minimal maintenance requirements, and quick recovery from potential issues.	**Robust Error Handling & Logging:** To identify and diagnose issues quickly. **Self-Healing Mechanisms:** (e.g., process restarts for software components). **Over-The-Air (OTA) Updates:** For timely bug fixes and feature

S.No	Characteristics	Description	Technology
			deployments without physical intervention. **Redundancy (Optional for critical applications):** Backup power supply.
5.	Performance	The solution prioritizes fast inference times to achieve high sorting throughput. This involves optimizing the machine learning model for edge deployment and efficient image processing.	**Quantized Models (e.g., TensorFlow Lite):** Reduces model size and speeds up inference on edge devices. **Hardware Acceleration:** Utilizing specialized hardware (e.g., TPUs, GPUs on Jetson, Coral Edge TPUs) for faster inference. **Optimized Image Processing Pipelines:** Efficient use of OpenCV functions. **Asynchronous Processing:** (If applicable) To handle image capture and inference concurrently.