

RecipeCart

Senior Design I — 60 Pages Draft



**UNIVERSITY OF
CENTRAL FLORIDA**

College of Engineering and Computer Science

Department of Electrical and Computer Engineering

Dr. Lei Wei & Dr. Chung Yong Chan

Department of Computer Science

Dr. Matthew Gerber & Mr. Richard Leinecker

EEL4914 – Group 35

Darren Ha — Computer Engineering

Quan Nguyen — Computer Engineering

Bryan Ha — Computer Science

Reviewers

Dr. Tanvir Ahmed — CS

Mr. Arup Guha — CS

Dr. [...] — ECE

Table of Contents

| | |
|---|-----------|
| 1.0 Executive Summary..... | 1 |
| 2.0 Project Description..... | 2 |
| 2.1 Project Context..... | 2 |
| 2.2 Project Goals..... | 2 |
| 2.3 Project Objectives..... | 3 |
| 2.4 Related Works..... | 4 |
| 2.4.1 Cust2Mate..... | 4 |
| 2.4.2 Samsung Smart Fridge and Family Hub..... | 4 |
| 2.4.3 Samsung Food..... | 5 |
| 2.5 Project Specifications and Constraints..... | 7 |
| 2.6 General Hardware Block Diagram..... | 8 |
| 2.7 General Software Flowcharts..... | 9 |
| 2.7.1 Food Object Detection Machine Learning Flowchart..... | 9 |
| 2.7.2 Mobile App Flowchart..... | 10 |
| 2.8 House of Quality..... | 11 |
| 3.0 Hardware Parts Comparisons..... | 12 |
| 3.1 Single Board Computers..... | 12 |
| 3.1.1 Raspberry Pi..... | 12 |
| 3.1.2 Google Coral Dev Board..... | 13 |
| 3.1.3 Nvidia Jetson Nano..... | 14 |
| 3.2 Camera Sensors..... | 15 |
| 3.2.1 Raspberry Pi Camera Module 3..... | 16 |
| 3.2.2 Raspberry Pi High Quality Camera..... | 16 |
| 3.2.3 Logitech C920e HD Webcam..... | 17 |
| 3.2.4 NexiGo N60 Webcam..... | 17 |
| 3.3 Weight Sensors..... | 18 |
| 3.3.1 HX711 Load Module + Load Sensor..... | 18 |
| 3.3.2 EK-2000i Compact Balance..... | 18 |
| 3.3.3 TLE Load Cell amplifier..... | 19 |
| 3.4 Power Supply Systems..... | 21 |
| 3.4.1 ZKETECH EBD-A20H..... | 21 |
| 3.4.2 Jesverty SPS-3010H..... | 21 |
| 3.4.3 XP POWER VCS50US12..... | 21 |
| 4.0 Software Comparisons..... | 23 |
| 4.1 Barcode Detection Software..... | 23 |

| | |
|---|-----------|
| 4.1.1 Dynamsoft Barcode SDK..... | 23 |
| 4.1.2 Zebra crossing C++ Barcode Decoder and DaisyKit Detector..... | 23 |
| 4.1.3 Web Barcode Detection API..... | 24 |
| 4.1.4 Self-Implemented OpenCV Python Program..... | 24 |
| 4.2 Food Image-Based Classification Software..... | 25 |
| 4.2.1 Google Cloud Vision..... | 25 |
| 4.2.2 Amazon Rekognition..... | 25 |
| 4.2.3 Meta DINoV2..... | 26 |
| 4.3 Recipe Recommendation System..... | 26 |
| 4.4 Recipe Generation System..... | 27 |
| 4.4.1 RecipeGPT..... | 28 |
| 4.4.2 RecipeMC..... | 28 |
| 4.5 Databases..... | 28 |
| 4.5.1. Ingredient Database..... | 28 |
| 4.5.2. User Database..... | 29 |
| 4.5.3. Model Database..... | 30 |
| 4.6 Backend Hosting Platforms..... | 30 |
| 4.6.1 DigitalOcean..... | 30 |
| 4.6.2 Firebase..... | 31 |
| 4.6.3 AWS Amplify..... | 32 |
| 4.6.4 Azure Mobile Apps..... | 32 |
| 4.7 Mobile Frontend Development Platforms..... | 34 |
| 4.7.1. Xamarin..... | 34 |
| 4.7.2. Flutter..... | 34 |
| 4.7.4. React Native..... | 35 |
| 5.0 Standards and Design Constraints..... | 36 |
| 5.1 Related Standards..... | 36 |
| 5.1.1 Video Packaging and Streaming Standards..... | 36 |
| 5.1.2 IEEE Federated and Shared Machine Learning Standards..... | 37 |
| 5.1.3 Network Communication Integration Standards..... | 38 |
| 5.1.4 Image Sensor Integration Standards..... | 40 |
| 5.1.5 Weighing Scale Integration Standards..... | 40 |
| 5.1.6 Power Efficient Systems Standards..... | 41 |
| 5.1.7 Software Development Standards and Best Practices..... | 41 |
| 5.2 Realistic Design Constraints..... | 43 |
| 5.2.1 Financial and Time Constraints..... | 43 |
| 5.2.2 Environmental, Social, and Political Constraints..... | 43 |
| 5.2.3 Health, Privacy, and Safety Constraints..... | 44 |

| | |
|---|-----------|
| 5.2.4 Manufacturability and Sustainability Constraints..... | 45 |
| 6.0 ChatGPT Applications and Limitations..... | 46 |
| 6.1 Large Language Model Historical Background..... | 46 |
| 6.2 ChatGPT Architecture..... | 47 |
| 6.2.1 Generative Pre-Training..... | 47 |
| 6.2.2 Reward Model..... | 48 |
| 6.2.3 Content Filter..... | 48 |
| 6.3 Analysis of Originality and Notable Limitations..... | 48 |
| 6.4 Prompt Engineering..... | 48 |
| 6.5 Potential Integration with RecipeCart..... | 49 |
| 6.5.1 Recipe Generation by Prompt Engineering..... | 49 |
| 6.5.2 Review Tagging System..... | 49 |
| 6.6 Recent or Related Developments of ChatGPT..... | 50 |
| 7.0 Hardware Design..... | 51 |
| 8.0 Software Design..... | 52 |
| 9.0 Implementation and Prototype Workflow..... | 53 |
| 10.0 System Testing..... | 54 |
| 11.0 Administrative Content..... | 55 |
| 11.1 Estimated Budget and Financing..... | 55 |
| 11.2 Project Timeline and Milestones..... | 56 |
| 11.3 Project Roles and Work Distribution..... | 58 |
| 12.0 Project Conclusion..... | 59 |
| Appendix A..... | 60 |
| References..... | 60 |

1.0 Executive Summary

1-2 pages here. Will complete at the end of report drafting

2.0 Project Description

2.1 Project Context

Consumerism has always dominated and shaped the modern global economy. In response to increasing consumption and demands, customer-centric companies often find themselves relentlessly upscaling supplies to meet consumer demands. However, demands do not directly translate to needs, and several supply chains often end up producing surpluses. The common consumer's spending habits are a microcosm of this exact development. A consumer's speculation of their needs often leads to them buying more supplies than necessary, increasing the ratio of unused resources going to waste.

Correcting spending behaviors would most effectively address this overconsumption problem but is not realistically achievable in the short term. Rather, the simpler approach would be to accommodate the current spending habits. Our project, the RecipeCart, attempts to encourage the usage of leftover resources in the kitchen by generating a collection of personally tailored recipes given an assortment of ingredients.

2.2 Project Goals

The primary goal of the RecipeCart is to simply reduce the waste of leftover ingredients by creatively repurposing them with novel recipe suggestions. To make the RecipeCart as user-friendly as possible, the RecipeCart will intelligently identify the ingredients being placed into its container and subsequently produce a list of user-relevant recipes. The classification will be done via an ensemble of pre-trained deep convolutional neural networks (CNNs). The recipes will be queried from an existing database, custom-filtered and sorted based on user needs, and displayed on a mobile platform for user interaction.

A more ambitious goal involves extending the recipe list to also include suggested grocery locations where the missing ingredients may be purchased. This development might entail bridging the RecipeCart with the Google Maps API and all relevant grocery store databases. Another advanced goal would be to integrate a scaling system into the RecipeCart to quantify the ingredients and suggest recipes that would more accurately reflect the state of the leftovers.

In designing a seamless input process for the end user, video streaming the food items should be strongly considered. Using images as input would imply having the user take pictures of the ingredients after they have been placed into the container. Video streaming the input would allow for dynamic addition and removal of select food items from the container without the user having to directly modify their collection. However, running machine learning models in tandem with a video-based input would pose an extreme challenge in both complexity and resource, making this feature a stretch goal.

The project would be incomplete without mentioning AI-generated recipes. Although generating creative and composite recipes through AI has already been done multiple

times, adding this feature would be a stretch goal because the resulting recipes do not necessarily guarantee a similar level of delicacy. Not only would the AI-inspired recipes need to be vetted for adequacy, some ingredients are simply incompatible and may result in a pungent smell or taste. Incorporating this feature would require a significantly more advanced generative model and also a system to determine the validity of the recipe.

2.3 Project Objectives

An ensemble of pre-trained deep neural networks balancing between object classification and barcode parsing would be implemented to accurately identify the input ingredient. The object classifying neural network would classify visually distinct ingredients based on vector encoding similarities. The second model would specifically detect for barcodes and query a designated barcode database for the food product metadata. Both neural networks would be deployed in tandem such that the object classification model is only invoked when the barcode detection model is unable to locate a barcode.

Given its heavy reliance on image input data, this project would feature a singular high-resolution camera to capture pictures of the food item from a central point of view. The camera would be mounted at the front of the container, pointing inwards at an appropriate tilt and would be connected to a pre-built, single board computer (SBC) for centralized power and control. The SBC itself would be equipped with the appropriate network adapters for easy communication with the server hosting the machine learning backend. The appropriate batteries and power sources would be embedded onto the sides of the container unit without excessive protrusion.

A wide-area scale would be embedded into the cart container to retrieve the quantity of a detected ingredient. The scale would be wired to the rest of the cart body and would send its results to the SBC for data cleaning and processing.

A server would be developed to host the ensemble learning model as well as the backend business logic to pre-process the input images and query the external ingredient and recipe databases. The server would also host its own database to maintain the basic client user metadata and also track common recipe selections, food preferences, and dietary needs to gradually tune the recipe recommendation engine.

The mobile user interface would feature a customizable catalog of all previously queried recipes, fronted by a basic login and sign up page. The app would allow the user to dynamically update their recipe preferences and also initiate a request to the main RecipeCart container to retrieve its current contents. The app would evidently include the function to generate recipe recommendations based on the retrieved ingredients.

A recommender system would be added to the recipe generation backend to provide more personalized recipe recommendations. The model will base its inferences on a variety of inputs such as recipe popularity, user ratings, recommendation histories, and frequently retrieved ingredients.

2.4 Related Works

The concept of the RecipeCart is not completely original and unique. There have been many prior inventions that attempt to facilitate the transition of food items from the grocery store to the home kitchen such as Amazon's Dash Cart and Samsung Food. This section highlights some related technologies that implement some form of food inventory management and tracking system, recipe database and personalization, and smart grocery shopping features.

2.4.1 Cust2Mate

Cust2Mate's Automated Smart Cart is a smart shopping cart with automatic item identification, weighing and checkout. These functions also attempt to address measures against theft or any error when taking items in and out of the cart (Lenovo, 2022). This feature greatly improves the users' shopping experience by omitting the waiting checkout time that regular buyers experience. Cust2Mate's carts are shelf-ready and operational, so they can be deployed quickly. The company currently administers pilots with several food chains around the world with the intention of significantly increasing their operations even further (Slater & Kreizman, 2022). The smart carts, as a launched product, are yielding significant returns on investments and gross profit.



Figure 2.1. Cust2Mate Smart Cart features

2.4.2 Samsung Smart Fridge and Family Hub

Samsung's advanced smart fridge integrates with other Samsung devices through the Family Hub and has AI assistance with Alexa. The fridge contains an inventory tracker, streaming services, and convenient, personalized apps such as memoing and shopping lists (Samsung, 2023). The Family Hub acts as more than a conventional smart fridge: it is a centralized medium for smart houses, integrating with other Samsung smart-home devices, as well as a computer itself with advanced user interface. The user interface on the most advanced Samsung smart fridge includes options to quickly view the contents

stored inside the fridge, instantly access saved recipes and videos, make meal plans based on current food inventory, and create and share grocery shopping lists.

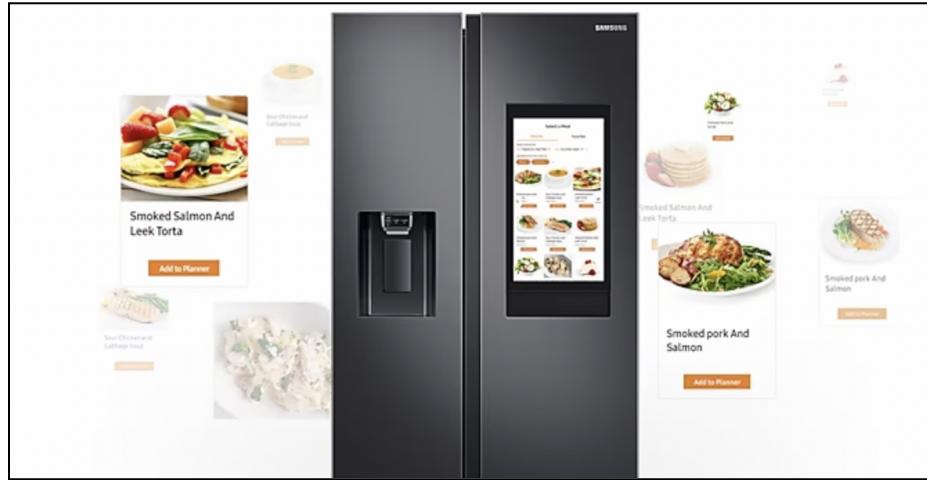


Figure 2.2. Samsung Family Hub integrated with the Smart Hub

2.4.3 Samsung Food

Samsung Food, originally branded as the Whisk app until August 2023, is a free recipe mobile app for recipe lookups, organization, and sharing. The app is integrated with many large and popular recipe websites such as the Food Network and AllRecipes.com [2.4]. The app initially acted as a meal planner, detailing users' daily view of their lunch, dinner, breakfast, and snacks, and subsequently outputting a health and wellness metric based on users' bioelectric impedance analysis (BIA). Certain advanced features—such as the ability to substitute ingredients in recipes and searching for recipes based on current ingredient inventory—were just recently added in 2023 to enhance users' experiences and engagement.

Users can search for recipes based on a variety of preferences and categories, ranging from international cuisines to specific dietary needs. Recipes can be further filtered based on preparation and cook time, and the ingredient quantities can be proportionally adjusted according to the desired number of servings [2.5]. Recipes are displayed to users on the home page based on their prior recipe preferences, ratings, and views. Samsung Food also automatically computes a health score for each recipe based on its nutritional contents per serving. Users can add missing ingredients to a shopping list and export it to online grocery store platforms such as Instacart or Amazon Fresh.

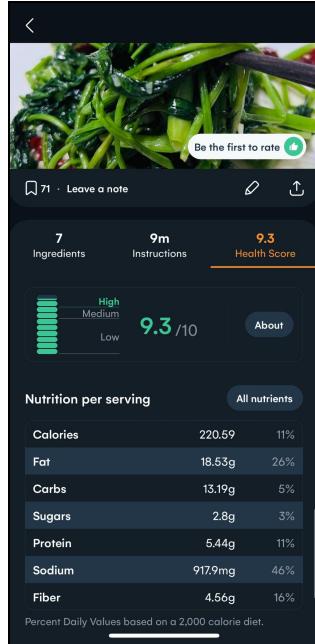


Figure 2.3. Example health score page for a recipe on Samsung Food

Samsung Food performs as a recipe-sharing and community-driven app, where users can follow popular food content creators, share comments, and assign approval ratings to recipes. Users can also create and join communities based on their food interests.

All three inventions exemplify successful launches of a product that eases a consumer's shopping and life experience by allowing them to monitor their purchases and their food inventory without the hassle of shuffling and inspecting through their cart or fridge. Cust2Mate designs a well-functioning smart cart with a simple user interface with a variety of virtual payment methods and omits the waiting time in the check-out lines. Samsung's Family Hub contains a built-in inventory tracker, as well as integration with different Samsung technologies such as Samsung Food, centralizing the fridge as one of the core smart appliances in a modern home.

Each technology provides a solution to a specific step along the meal preparation "pipeline," but none of them is fully encompassing or end-to-end. Our project aims to implement inventory tracking features as seen in Cust2Mate and Samsung's Family Hub, subsequently allowing the user to explore different recipe ideas based on their current ingredients as inspired from Samsung Food.

The RecipeCart is intended as an add-on to the existing technology, accommodating to user dietary needs in a seamless manner. It attempts to provide a pipeline from the raw, physical ingredients to digitally manageable and customizable recipes. The RecipeCart also attempts to address a shortcoming that none of these technologies seem to properly resolve: food object detection and classification.

2.5 Project Specifications and Constraints

| Specifications | Description | Value |
|------------------------------------|--|--------------|
| Object Classification Time | Time to identify the object and return a list of ingredients | < 5 seconds |
| Object Classification Accuracy | A measure of the ML model's performance | > 70 % |
| Recipe Recommendation Time | Time to return a list of recipes given user preferences and a list of identified ingredients | < 10 seconds |
| Minimum Number of Recipe Outputs | Number of recipe recommendations guaranteed to generate | > 1 recipe |
| Absolute Weight Error | A measure of the tolerated error margin from the scale | < 5 grams |
| Weight and Quantity Retrieval Time | Time to calculate and obtain weight and quantity of an ingredient | < 2 seconds |
| Broadcast Delay | Time to broadcast output from hardware to the server | < 2 seconds |
| Product Weight | Weight of the hardware | < 4 kg |
| Power Consumption | Power intake of hardware | < 40 Watts |
| Battery Life | Battery life before next recharge | ~ 4 hour |
| Budget Management | Total estimate price of the whole production process | < \$500 |

Table 2.1. Engineering design specifications

2.6 General Hardware Block Diagram

The hardware architecture of the RecipeCart can be dissected into three main groups of components: the power supply system, the sensors, and the single board computer unit. The camera and weight sensors would be powered through the SBC's power module, which gets its own power from an integrated power jack. The microcontroller would be readily integrated with all the necessary modules to function as a single board computer. The sensors would be wired to the microcontroller unit to forward their observations. The microcontroller would pre-process all observed data before forwarding the request to the backend server via a Network Interface Card (NIC) inside the Wi-Fi module.

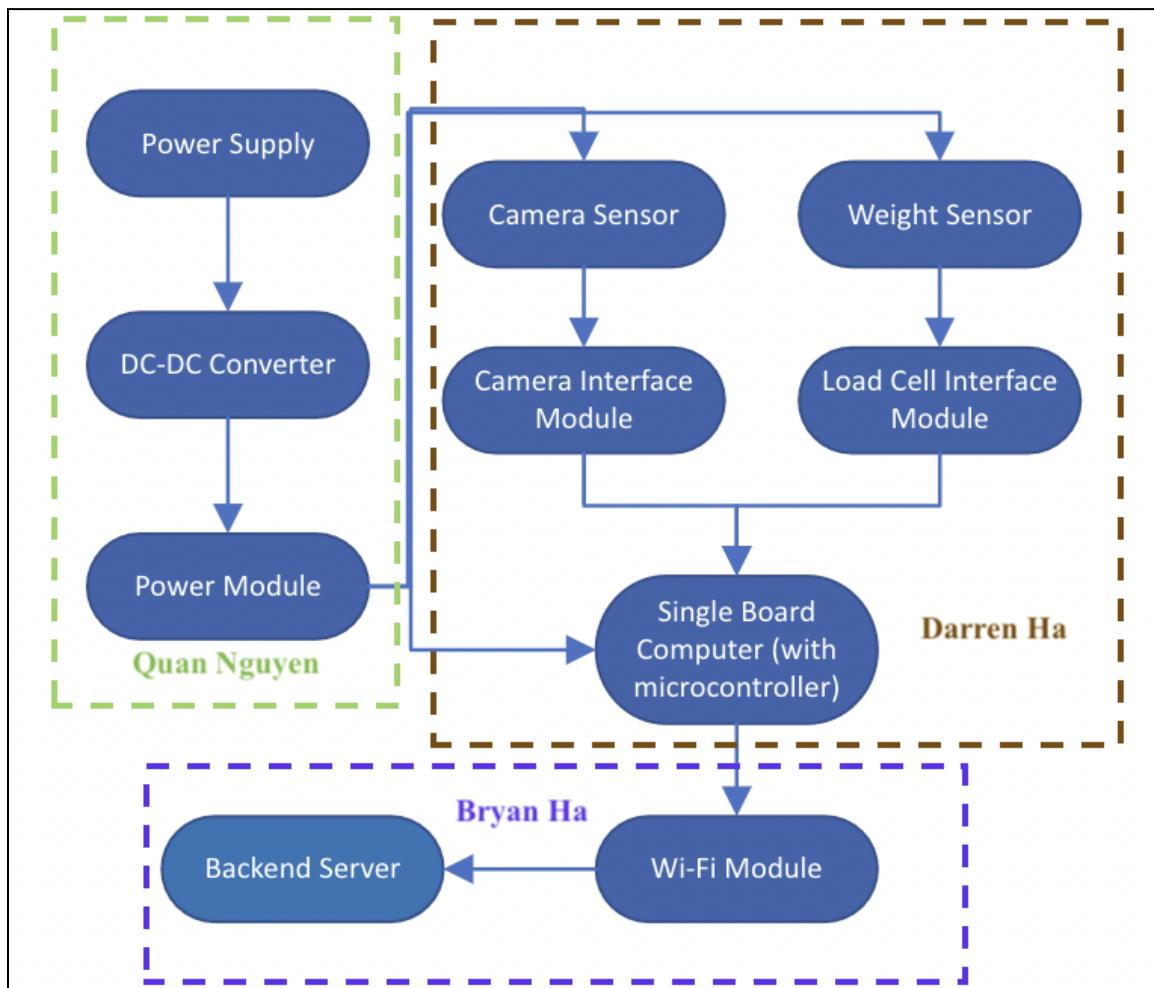


Figure 2.4. General hardware block diagram

The weight scale platform must be placed at an elevated level, separate from the storage area. The camera must be positioned at an angle such that it could holistically capture the contents placed onto the weight scale platform, without peeking views into the storage area.

2.7 General Software Diagrams

2.7.1 Ingredient Detection and Classification Flowchart

The below diagram depicts the general process of the ingredient detection and classification logic. Given the two methods of identifying an ingredient, the process is split into two primary pipelines: one for barcode querying and the other for an image-based food classification. The barcode-based classification is conditional on the detection of a barcode, while the food object classification method is triggered upon receipt of a weight value from the integrated scale platform.

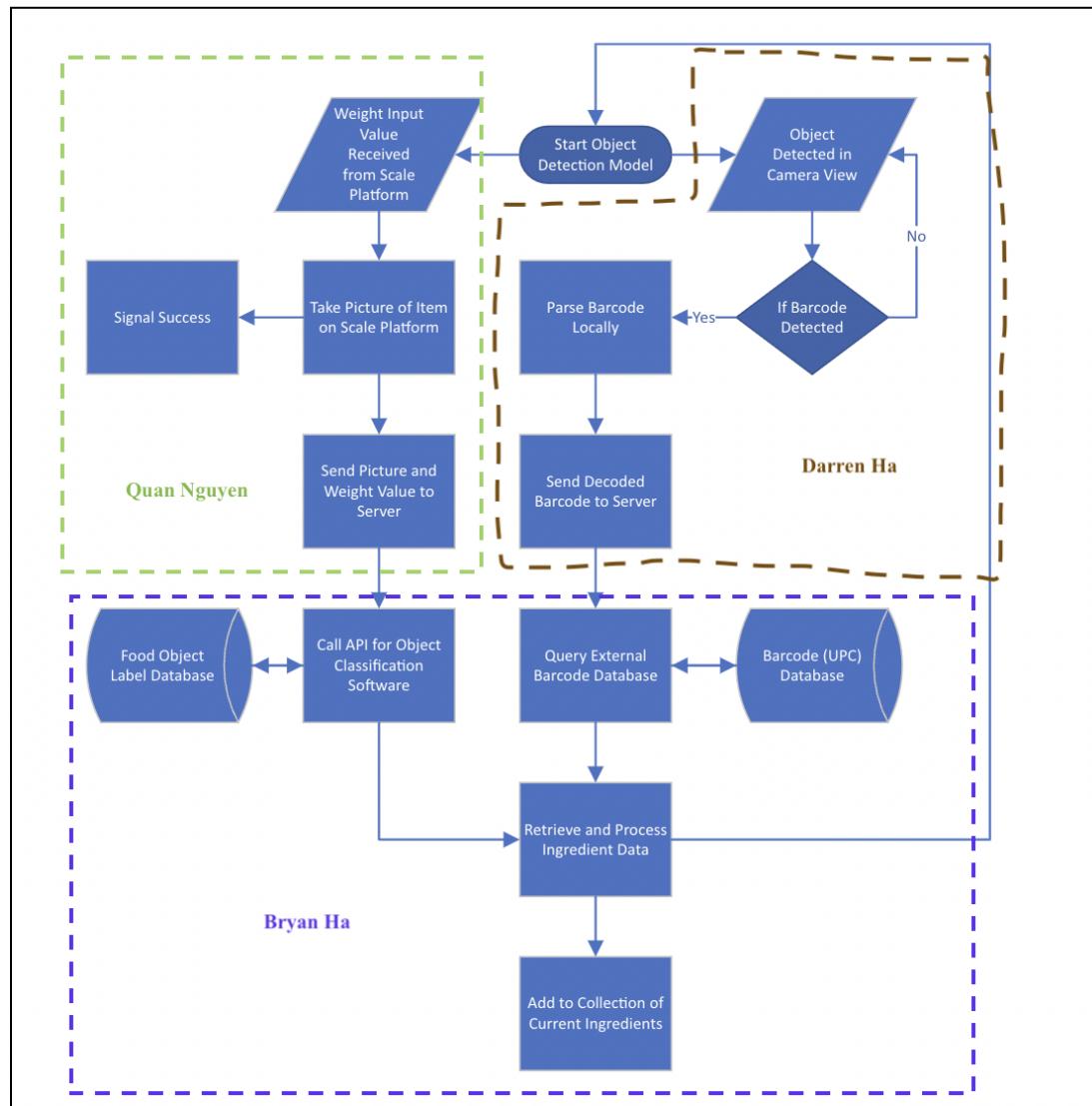


Figure 2.5. Ingredient Detection and Classification Flowchart

2.7.2 Mobile App Use Case Diagram

The RecipeCart would have a mobile app that allows signed-in users to connect to the dedicated containers; edit their diet, allergen, and preference profiles; view their ingredients; and generate and save recipes. The generation or search of recipes would employ a combination of database information and an ML model, which would be presented to the user frontend through the API.

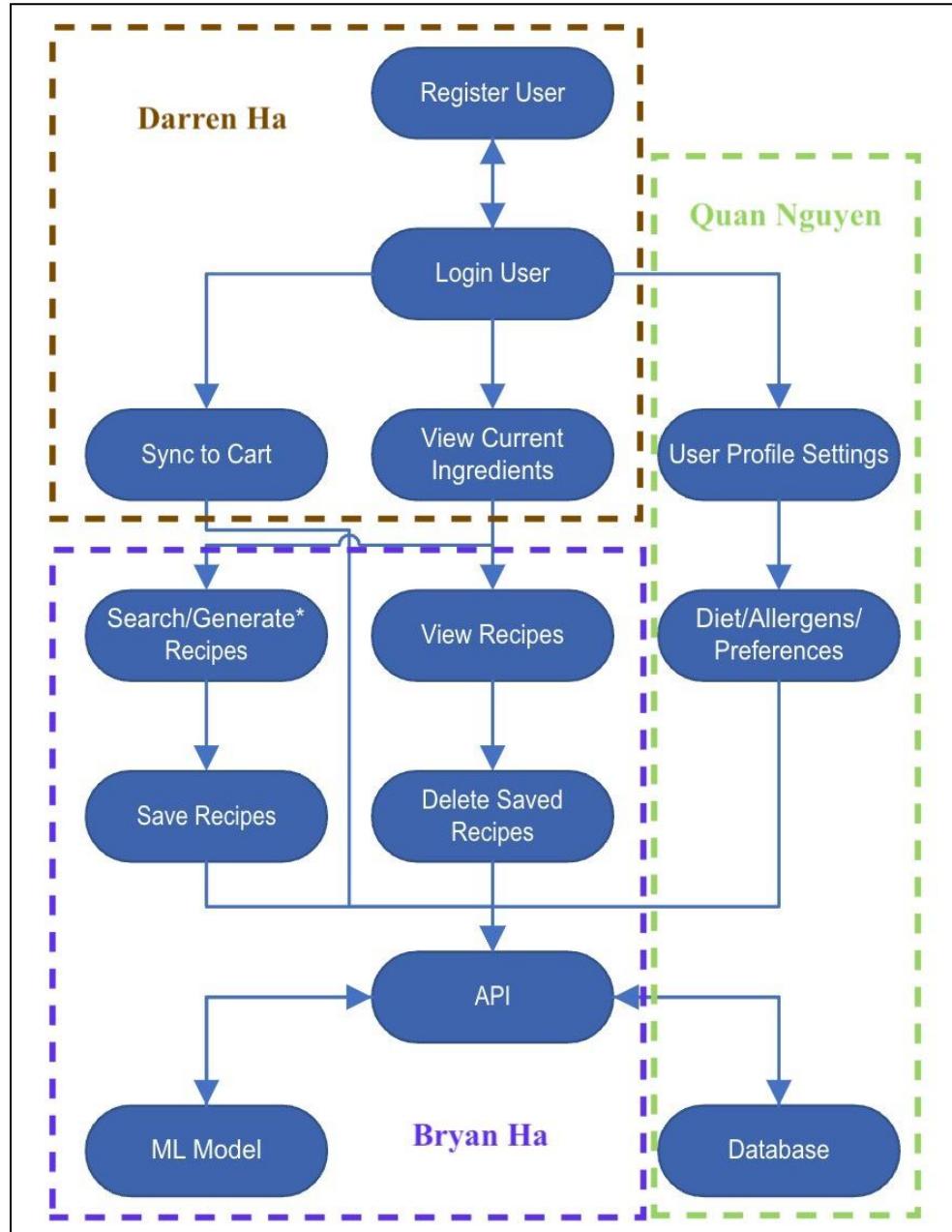


Figure 2.6. General mobile app use case diagram

2.8 House of Quality

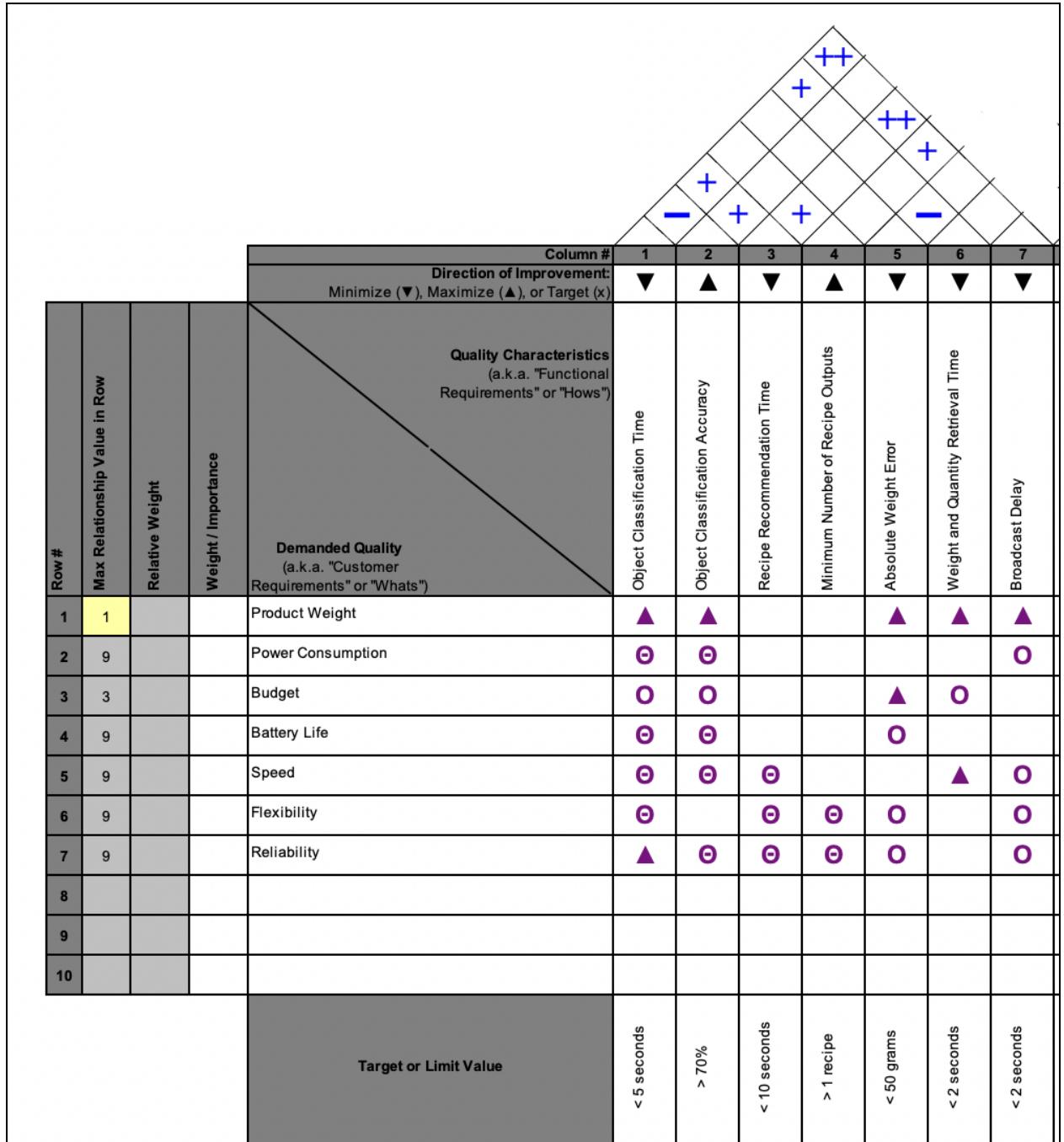


Figure 2.7. House of Quality Diagram

The house of quality shows the main areas that should be focused on to maximize consumer satisfaction and its relation to engineering requirements. Analyzing the figure above, the project should focus on maximizing the operational efficiency of the ingredient recognition and minimizing power consumption and expenses.

3.0 Hardware Parts Comparisons

Following the general hardware architecture described in Figures 2.1, this chapter examines several hardware technologies and their potential integration with the project's principal pipeline. While most parts highlighted are cross-compatible with one another, we discover that certain parts require extensive costs and efforts to integrate with others and may lead to unnecessary software complications. Parts are compared based on ease of integration, complexity, flexibility, cost, and also their adherence to the project specifications and design constraints.

3.1 Single Board Computers

The microcontroller is the heart of any embedded system and operates as the link between the hardware sensors and the software-side business logic. Typically, the microcontroller exerts central computational control over the various modules of an embedded system [3.1]. Its purpose is to perform efficient edge computing without the involvement of an actual server. In the context of the RecipeCart, we examine two approaches at using the microcontroller, both of which entail having it pre-made and integrated onto a single board computer (SBC). The first method involves the SBC as a simple edge device that captures, lightly pre-processes, and relays measurements to the server, where all the machine learning heavy-lifting is done. The latter method features the SBC as a more independent mini-computer capable of running some basic machine learning inferences—such as real-time barcode or text detection—before sending all of the data to the ML backend. The latter method calls for a well equipped and performant SBC while the former implies a less power consuming and simpler SBC.

3.1.1 Raspberry Pi

The Raspberry Pi Family provides powerful mini-computers that can run operating systems, browse and interact with the internet, and perform edge computing on IoT devices. The Raspberry Pi 4 is the most well-supported, cheapest option among the SBC options considered, while still maintaining the ability to perform basic machine learning inferences. It has for CPU the Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC, with a clock rate of 1.8 GHz [3.2]. The Raspberry Pi 4 is available at different memory sizes with varying prices. It comes with numerous USB and micro-HDMI ports and supports wireless LAN, Gigabit Ethernet, and Bluetooth connectivity. It also has a 2-lane MIPI CSI camera port.

Its main drawback is its weaker graphic processing power relative to the other SBCs, which may limit its machine learning applications depending on the deployed model and the inference type and frequency. The Raspberry Pi 4 can be paired with the Coral USB Accelerator to enhance its machine learning capabilities [3.3]. However, this addition would not only significantly ramp up the total cost of the SBC but also risk overheating

the central module. Running machine learning models for an extended time period on the Raspberry Pi 4 would require installing and configuring an additional heatsink and fan.

The Raspberry Pi 5 is set to be released at the end of October 2023 and is much higher performing than its older counterpart. It comes with a better CPU, the Broadcom BCM2712, 64-bit quad-core Cortex-A76 ARM with integrated L2 and L3 caches. The Raspberry Pi 5 also includes VideoCore VII GPU, allowing it to run machine learning workloads significantly more efficiently and faster than the Raspberry Pi 4 [3.4]. While the Raspberry Pi 5 has an integrated fan for its cooling purposes, running continuous and compute-heavy workloads would still require an additional active cooling system.

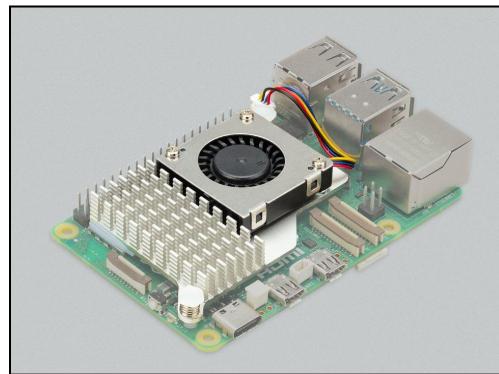


Figure 3.1. Raspberry Pi 5 with integrated fan

Given the novelty of the Raspberry Pi 5, it has relatively sparse documentation and community support, making it potentially more challenging to navigate and integrate than the Raspberry Pi 4.

Selecting the Raspberry Pi for the RecipeCart would be best suited for lighter machine learning workloads being performed on the edge. It would imply having most of the machine learning processing and inferencing occurring in the server-based backend.

3.1.2 Google Coral Dev Board

The Google Coral Dev Board is the most expensive and high performing option out of the bunch. Its CPU is the NXP's i.MX 8M SoC, with quad-core Cortex-A53 and Cortex-M4F—a lower power microcontroller designed to interact with external sensors. Not only does it have the Integrated GC7000 Lite Graphics as its GPU, it is also integrated with Google proprietary Edge TPU coprocessor and is capable of executing 4 trillion operations per seconds (TOPS), consuming 0.5 Watts per TOPS [3.5]. In addition to various USB ports and peripheral connections, it also supports 5 GHz WiFi via a 2x2 MIMO, Giga Ethernet, and Bluetooth 4.1. Video integration is supported through an HDMI 2.0a port and 4-lane MIPI-CSI2 camera FFC connector.

The Coral Dev Board's main flaw is its lack of compatibility with other deep learning libraries and frameworks [3.3]. Given its TPU module, it is specifically tailored to quickly run TensorFlow-based workloads, with limited flexibility and support provided for other deep learning software.



Figure 3.2. Coral Dev Board

The Coral Dev Board would be most appropriate for intensive on-the-edge machine learning applications. In addition to restricting the machine learning framework to only TensorFlow, incorporating it into the RecipeCart would imply having it process most, if not all, the workloads with little reliance on the backend server.

3.1.3 Nvidia Jetson Nano

Nvidia's Jetson Nano is the smallest form factor available as part of the Jetson Family of AI-dedicated SBCs for optimized edge computing. It is much more flexible than the Coral Dev Board, all the while being more powerful than the Raspberry Pi 4 and 5. The Jetson Nano supports not only TensorFlow but also PyTorch, Keras, MXNet, and scikit-learn among others [3.3]. It also has many SDK libraries and extensive community support. It boasts a quad-core ARM A57 with a clock rate of 1.43 GHz as CPU and is integrated with Nvidia's 128-core Maxwell GPU [3.6]. This SBC comes with 12 lanes of MIPI CSI-2 for camera sensors, several USB connectors, and its own active cooling system.

While the Jetson Nano is less efficient than the Coral Dev Board when data processing and inferencing, its increased compatibility allows it to have a wider range of applications. Enabling WiFi and Bluetooth also requires an external dongle and additional configuration as the factory unit only comes with support for Gigabit Ethernet.



Figure 3.3. Jetson Nano

The Jetson Nano is by far the most flexible and balanced option of the trio as it provides adequate compute power to perform the necessary machine learning workloads on the edge while maintaining compatibility with a variety of deep neural network frameworks.

| Name | Raspberry Pi 4 / 5 | Coral Dev Board | Jetson Nano |
|-------------------------------|---------------------------------------|---------------------------------------|--------------------------|
| CPU | Quad-core ARM Cortex-A72 / Cortex-A76 | Quad-core ARM Cortex-A53 + Cortex-M4F | Quad-core ARM Cortex A57 |
| GPU | None / VideoCore VII GPU | Integrated GC7000 Lite Graphics | 128-core Nvidia Maxwell |
| TPU | None | Google Edge TPU coprocessor | None |
| Memory | 1, 2, 4, 8 GB / 4, 8 GB | 1, 4 GB | 4 GB |
| WiFi Module | Included | Included | Excluded |
| Power Consumption | 3-10 W | 6-12 W | 5-10 W |
| Cooling Module | Excluded | Included | Included |
| Software Compatibility | Most ML libraries | Only with TensorFlow Lite | Most ML libraries |
| Community Support | Extensive / Sparse | Good | Extensive |
| Price | \$ 60-80 / \$ 100-120 | \$ 160 | \$ 147 |

Table 3.1. Single Board Computer comparisons

3.2 Camera Sensors

The concept of the RecipeCart revolves around the ability to effortlessly and accurately identify food items and ingredients based on real-time video input. Ingredient classification would be done through a combination of barcode parsing, text analysis, and image recognition. Since the desired camera sensor must be able to clearly capture the barcodes and printed texts, video quality and performance are highly prioritized. Integration complexity is also strongly considered. We thus examine two classes of camera sensors: the native CSI bus camera modules and the USB cameras.

Given their direct integration and proximity to the SBC, the native CSI camera modules are optimized for computing and memory usage. They are generally smaller, provide low level control, and are highly customizable, allowing for hardware performance maximization.

The USB cameras are more high level and are thus easier to mount and integrate with the SBC; conversely, they offer less control and room for hardware performance optimization [3.7]. Since they operate over the USB bus, they are more subject to latency and consume more CPU time. USB cameras are also more expensive than the native CSI camera modules.

3.2.1 Raspberry Pi Camera Module 3

The Raspberry Pi Camera Module 3 is the newest model in the Raspberry Pi CSI-2 camera series. Its 12-megapixel Sony IMX708 image sensor allows it to record full high definition videos at 50 fps with a very high signal-to-noise ratio [3.8]. The original model has a diagonal field of view (FOV) of 75°, while the wide model is capable of expanding to 120° at the cost of a slightly worse depth resolution. The Camera Module 3 also has an auto-focus feature, allowing for dynamic depth adjustments. The lens on the Camera Module 3 are fixed and cannot be switched out.

The Camera Module 3 produces a slightly distorted far-range view when exposed to intense outdoor lighting. However, this discrepancy can be ignored given that the selected camera sensor for the RecipeCart will primarily be used for close-up views. The Camera Module 3 is subject to “exposure hunting” where it sometimes switches between a lighter and darker tone in searching for the right level of exposure.

3.2.2 Raspberry Pi High Quality Camera

The Raspberry Pi High Quality Camera Module is significantly heavier and bulkier than the Camera Module 3 but has a higher still resolution at 12.3 Megapixels. Its video resolution is worse with 2028 x 1080 pixels at 50 fps. It uses the Sony IMX477 as its camera sensor and comes with a lens mount, allowing for different lens options based on the C/CS- or M12- interfaces and an adjustable field of view [3.8]. The HQ Camera was originally designed as a higher performing alternative to the Camera Module 2. As such, in contrast to the newer Camera Module 3, the HQ Camera offers little to no added benefits in cost efficiency. The HQ Camera produces a cooler, bluer tone whereas the Camera Module 3 produces a slightly warmer, albeit darker view [3.9]. With a 6 mm wide angle lens, the HQ Camera has a variable aperture and a manual focus mechanism.

While the HQ Camera does not suffer from exposure hunting, the mounted lenses primarily determine the quality of the captured footage; a low quality lens would evidently produce low quality videos.

| Name | Camera Module 3 | HQ Camera |
|-------------------------|------------------------|------------------|
| Still Resolution | 11.9 Megapixels | 12.3 Megapixels |
| Video Resolution | 2304 x 1296p56 | 2028 x 1080p50 |
| Sensor | Sony IMX708 | Sony IMX477 |
| Focus | Motorized (Auto) | Adjustable |
| Field of View | 75° | Adjustable |
| Price | \$ 25-35 | \$ 50 |

Table 3.2. CSI-2 camera module comparisons

3.2.3 LogiTech C920e HD Webcam

The LogiTech C920e HD Webcam is a USB camera that has a max video resolution of 1920 x 1280 pixels at 30 fps. It has a diagonal field of view of 78° and provides automatic focus [3.10]. Compared to the Raspberry Pi camera modules, the C920e webcam is easier to configure and mount but has lower resolution and field of view. This USB camera option offers a balanced set of features at a relatively low cost and with little setup.

3.2.4 NexiGo N60 Webcam

The NexiGo N60 Webcam outputs videos at equal resolution to the LogiTech C310 with 1920 x 1080 pixels at 30 fps. It has a much wider diagonal field of view of 107° and uses a 1/2.9-inch CMOS digital image sensor allowing it to adequately define objects between 20 and 118 inches from the camera [3.11]. Again, the NexiGo N60 Webcam offers high level control and ease of integration at the expense of lower image quality and focus adjustability.

| Name | LogiTech C920 HD Webcam | NexiGo N60 Webcam |
|-------------------------|--------------------------------|--------------------------|
| Video Resolution | 1920 x 1080p30 | 1920 x 1080p30 |
| Field of View | 78° | 107° |
| Focus | Auto | Fixed |
| Price | \$ 58.90 | \$ 29.99 |

Table 3.3. USB camera module comparisons

3.3 Weight Sensors

3.3.1 HX711 Load Module + Load Sensor

The HX711 is a load module that will be used as an amplifier and ADC, connected to the load cell and microcontroller to work as a digital scale used to measure products [3.12]. The load cell's capacity can be determined depending on the upper-limit weight that we decide to perform our software on.

The HX711's library has been officially tested under RISC-based microcontrollers and wifi microchips, as well as community-tested on the Arduino and Jetson Nano [3.12]. It is positive to assume that the load module's library will be compatible with most microcontrollers.

The HX711 and the loading cells are affordable and simple to use. It has been around for a while so there are many community forums and official support regarding the module, as well as many projects involving the module's usage. As a result, it is easy to set up the circuit and calibrations, as well as data collecting and broadcasting.

One problem that might occur during the process of developing the sensor is that there is a history of unstable, fluctuating value reports due to temperature affecting the performance load cells. In addition, there may be value drifting errors due to the circuit being sensitive to supply voltage or other mechanical issues [3.13]. Therefore, implementing the weight sensor using this approach should be handled carefully with precision.

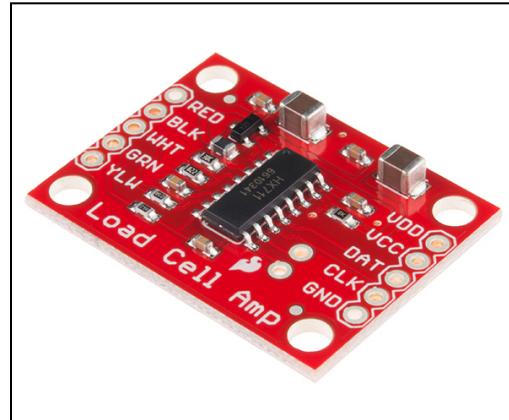


Figure 3.4. HX711 Load Module

3.3.2 EK-2000i Compact Balance

The EK-2000i, as a laboratory scale, greatly simplifies data collection through USB ports or similar converters; we can easily collect data onto our computer for simple calculations.

The scale has an FTDI RS to USB converter, which can be fed into any Raspberry Pi microcontroller for data collection [3.14]. This process should be similar for any laboratory scales. Since this allows for easy calculations, no external libraries are needed to acquire weights from the scales, and we can program straight off data collected into the microcontroller.

The balance allows accurate data collection with low risk of errors and hardware complications. In addition, since it doesn't require any external libraries, it is flexible, and can work with any microcontroller as long as it has the means to feed data into the system.

However, the balance is on the pricey side, with the cheapest unit being \$328 [3.14]. The scale is also meant to weigh lighter objects for micro-experiment purposes. Therefore, it is hard to "scale" the project (as in measure heavier products). Another problem is calibrating the scales. Since most lab scales have manual zero buttons, it is hard to zero out the weights automatically whenever the objects are put out of the platform.



Figure 3.5. EK-2000i with FTDI Input

3.3.3 TLE Load Cell amplifier

The TLE load cell contains a built-in analog, with an LCD display for voltage and current [3.15]. This cell replaces more simpler load modules while allowing more ports for parallel load cells and simplifies the troubleshooting process by showing currents and voltage without having to use the electrician's multimeter.

The TLE contains an RS485 port that can be directly connected to any PC's serial port for data extraction [3.15]. For microcontrollers listed above, a serial port to usb adapter is needed. There is no library needed to operate the TLE to obtain and transfer data from the cells to the microprocessor. The amplifier also contains four load cell slots, which can be used to measure multiple items or heavier items.

The TLE is designed to be very user-friendly, with built-in display of volt input and analog outputs, so it is easy to set up and troubleshoot the system. The TLE contains several ports for parallel load cells implementation that works well for extracting weights in a bigger scale. It is also mountable, which helps with pre-built electrical panel enclosures.

The TLE is in the expensive range, with the cheapest product found at \$190.00. In addition, the additional ports are unnecessary in our projects, where we only need one port to measure the weight of food inventory.



Figure 3.6. TLE Load Cell Amplifier

| Name | HX711 | EK-2000i | TLE |
|--------------------------------------|------------------------------------|---|-----------------------------------|
| Microcontroller Compatibility | Adapter needed | Native Compatibility | Adapter needed |
| Programmable | Yes | No | No |
| Community Support | Good / Extensive Developer Support | Sparse Developer Support | Sparse Developer Support |
| Additional Resources | Load cell / gauge | None | Load cell |
| Power Consumption | 1.5 mA | <i>Not Specified</i> - 250 hours battery life | 0-10VDC or 4-20mA output versions |
| Price | \$10.95 | \$328.00 | \$190.00 |

Table 3.4. Load sensor / module loadout comparisons

3.4 Power Supply Systems

RecipeCart would need a power supply that could provide adequate wattage to the hardware components while minimizing weight and price. In order to adhere to the design constraint of a maximum total product weight of 4 kg and a maximum power consumption of 40 Watts, the supply should be at most 1kg - 1.5 kg and generate at least 45 Watts.

3.4.1 ZKETECH EBD-A20H

The EBD-A20H is a hybrid DC power supply / load tester that can produce up to 200W under monitor. Its' settings can go up to 30.0V and 20.0 A. The supply contains a DSC-CC test mode, with voltage, current and capacity test within 0.5% error [3.16]. To monitor this, it contains an EB software to display the relevant test information. The supply is great for testing out the product, as well as monitoring hardware implementations and benchmarks. As a power supply, it provides more than enough to support the hardware components. It is on the more expensive side, and the hardwares don't need up to 200W to function properly

3.4.2 Jesverty SPS-3010H

The SPS-3010H is a DC power supply that can produce up to 30V and 10A, including encoder knobs and LED display that provides readings of power. Aside from the grounding and sign ports it also contains a 5V/2A USB port for charging separate components if needed [3.17]. The SPS-3010H is in the cheaper range compared to other DC power supplies. It is also light, and contains silent internal fans, perfect as an add-on to consumers' devices. Its' main weaknesses are that the capacitor remains discharging for a long time after turning the supply off, and that it supplies peak voltage on launch, causing some potential safety issues.

3.4.3 XP POWER VCS50US12

The VCS50US12 is an AC-DC converter with adjustable output and universal input that can produce up to 12V/4.2A. It does not contain a battery, and purely functions as a power converter [3.18]. The VCS50US12 is light and it does not have any extra function other than performing AC/DC conversion and power clipper, so it needs a connection to a bigger external power source such as a wall plug. It can function between -25C to 75C [3.18], great for RecipeCart as an addon to smart fridges.



(1)

(2)

(3)

Figure 3.7. (1) EBD-A20H Power Supply. (2) SPS-3010H Power Supply. (3) VCS50US12 Converter

| Name | ZKETECH EBD-A20H | Jesverty SPS-3010H | VCS50US12 |
|---------------------|------------------|--------------------|-----------|
| Voltage draw | 30 V | 30 V (5V USB) | 12 V |
| Current draw | 20 A | 10 A (2A USB) | 4.2 A |
| Power Draw | 200 W | 300 W | 50 W |
| Software | Yes | No | No |
| Weight | 8 kg | 1.10 kg | Ignorable |
| Price | \$79.00 | \$47.99 | \$22.23 |

Table 3.5. Power supply comparisons

4.0. Software Comparisons

Building on the software diagram in Figure 2.4, this chapter describes available software that may be used towards implementing the core features of RecipeCart. The different components are examined based on their scalability, efficacy, and cost of service. Moreover, we prioritize systems that can be run locally on a microcontroller at low computational costs over dependence on server computation resources.

4.1 Barcode Detection Software

Barcode detection is the simplest and most efficient method of recognizing ingredients. It is used in stores to check out items, and in smart fridges to keep track of resources. Since the content of the barcode contains an exact product number, it allows for precise and consistent classification of items simply by accessing a database containing the product number. Barcodes must adhere to three rules: it must be unique to each product; it must be permanently associated with the product it is assigned; and it needs to be random to ensure there is enough capacity for future products. As such, it is the preferred method of classifying ingredients for RecipeCart. Barcode detection should be the primary method of detecting and identifying ingredients due to its low error rate and high operational efficiency. As a downside, leveraging barcode detection will require access to a database containing all the product numbers and their associated information, many of which will incur a cost.

4.1.1 Dynamsoft Barcode SDK

The Dynamsoft Barcode SDK is a pretrained barcode detection model implemented in C/C++ for speed in ARM64 processors. Dynamsoft Barcode SDK works by extracting the boundaries, cropping, downscaling, grayscaling, transforming, and then extracting the barcode. It provides quick and easy access to a barcode scanner and removes the need to personally train a barcode scanning model and works well with streamed data. [4.1] Since this is an enterprise product, it performs extremely well and is optimized for efficiency and accuracy. This service requires a license for use with a 30-day free trial, after which will incur a cost about \$100 per month. This may be out-of-budget for the product prototype of RecipeCart, but may become a more viable option if we decide to make a real product.

4.1.2 Zebra crossing C++ Barcode Decoder and DaisyKit Detector

ZXing is an open-sourced kit for barcode decoding on image data. This can be used in tandem with an open-source barcode detector like DaisyKit. It does not incur a subscription cost due to being open-sourced. Both the detector and decoder have been implemented in C++ to maximize computational efficiency [4.2]. ZXing and DaisyKit are powerful options for barcode detection and processing, it incurs no cost to the project and is currently the most favored option among the different available technologies.

4.1.3 Web Barcode Detection API

Firefox and Google offer a barcode detection and scanning API, using it would require an active connection with the API and this may incur time costs due to requiring connection to an external server. This option is not open-sourced and will still require a picture or video input for barcode processing. This might result in unfeasible connectivity dependence on the server and result in high traffic. Due to the dependence on an external server to detect and scan barcodes and data streaming, the resolution of the camera will need to be minimized and this may impact accuracy. Compared to options like ZXing and Dynamsoft, which provide means to run detection algorithms locally, web-based barcode detection algorithms increase the dependence on external services which can only incur time delays.

4.1.4 Self-Implemented OpenCV Python Program

Another option is to program the detection algorithm using widely available libraries like OpenCV, and extract the barcode. This option is freely available but since Python is noticeably slower than any compiled implementation in C/C++, there may be performance decrease. Furthermore, this option requires more work than other existing options like the ZXing library. Therefore, the OpenCV option should be left open for other more sophisticated image processing.

| Name | Dynamsoft | DaisyKit | Web API | OpenCV |
|---------------------------|---------------------|-----------------------------|---|-----------------------------|
| Price | \$100 / month | Free | Free | Free |
| Implementation complexity | Product ready to go | Product ready to go | Coding detector and requests | Coding detector and decoder |
| Speed | Enterprise-tier | Slow detector | Slow decoder | Slow overall |
| Performance | High accuracy | May not detect deformations | Decreased quality due to data streaming | May not detect deformations |

Table 4.1. Barcode detection software comparisons

4.2 Food Image-Based Classification Software

Another component of RecipeCart's ingredient detection algorithm running in tandem with the barcode detection is the image classification module, which is intended to classify items whose barcodes are not in sight or deformed. Training a scalable and robust classification model is essential to ingredient detection. The image classification module is expected to run on a server level to enable more powerful models. An ideal classification model would be able to learn to identify new ingredients quickly.

The naïve option is to simply train a model that can recognize and classify the different products. ResNet is a simple model for image classification combining Convolutional Neural Networks with residual layers to ensure limited loss of information and allows for state-of-the-art empirical performance.

However, this method is infeasible since there are thousands if not millions of different products and variations. Furthermore, having a larger number of classes does not scale well with standard classification networks. And for every new class/ingredient, the model will need to be retrained which is extremely costly.

Instead, we can train ResNet as a general encoder model that will store latent embeddings for different items and use another method for classifying the encoded data. This will allow us to do scalable reverse search on items that have been detected.

Due to the nature of standard classification AI, simply training a regular classifier will not allow for adding new unseen products without retraining. As such, we prioritize algorithms that decouple the number of classes and the training of the detection algorithm. One popular method in industry for few-shot learning is to perform a similarity search with other learned classifications and then adopting the classifier of the k-nearest neighbors which will allow for few-shot classification and scalability with little to no retraining. [4.4]

4.2.1 Google Cloud Vision

Google Cloud Vision is Google's computer vision pretrained model. It is able to identify products from images and has strong performance as a few-shot classification model. It can easily integrate with other Google services like BigQuery and Cloud Functions for end-to-end Google-based implementations. Google Cloud Vision can work on a wide variety of image data including PNG, JPG, GIF, BMP, Raw, WebP, etc. Google Cloud Vision also allows for users to use a free trial version which is free and allows users to determine whether they wish to use the product for their intended use case. [4.5]

4.2.2 Amazon Rekognition

Amazon Rekognition is Amazon Web Service's cloud-based pretrained computer vision model. Like the Google Cloud Vision, it is able to perform few shot classification and can easily integrate with other Amazon products like AWS S3 and AWS Lambda for end-to-end pipeline implementations [4.6]. Unlike Google Cloud Vision, Rekognition

only works on PNG and JPG data. While images are processed on a per use basis, Google's pricing is slightly more than that of Amazon's. [4.7]

4.2.3 Meta DINOv2

DINOv2 is Meta's pretrained computer vision model. It is able to detect depth, segmentation, and instance retrieval at high accuracy with few-shot capabilities. In addition it is open-sourced and can be implemented and integrated on the user end [4.8]. As such it does not incur any costs to the user aside from server maintenance. When combined with a Vector Database for similarity search, users may be able to perform efficient search and detection of products.

| Name | Google Cloud Vision | Amazon Rekognition | Meta DINOv2 |
|----------------------|---|---|--|
| Flexibility | Classification, Image Segmentation, Image Tagging, etc. | Classification, Image Segmentation, Image Tagging, etc. | Object Detection, Image Segmentation, Image Encoding, etc. |
| Price | First 1000 / month free, \$ 0.0015 / month after | First 5000 / month free, \$ 0.001 / month after | Free |
| Effectiveness | Self-Contained | Self-Contained | Will Need VectorDB |

Table 4.2. Image classification software comparisons

4.3 Recipe Recommendation System

At the core of RecipeCart's algorithm is a recipe recommendation system. The recipe recommendation is expected to tailor and recommend recipes to users that match their user profile and diet. One major consideration is that food apps have historically little feedback from the same user, often leading to a lack of potentially insightful data. Users are not necessarily inclined to like same-tasting food and can get sick of things. An ideal recommendation system should form malleable profiles of each user and recommend a variety of different foods.

Existing methods of recommendation systems are based on similarity metrics, often leveraging similarity between users for recommendation of items or similarity between liked products. These systems can fall into two major families: content-based filtering which recommends based on a user's preference history and recommends similar products, and collaborative filtering which leverage user profiles to recommend products that other similar users have liked. Content-based filtering methods benefit from a good

embedding of recipes such that similarly preferred recipes are embedded close together while incompatible ones are repelled. Content-based filtering is susceptible to recommending similar recipes which may get repetitive for users [4.9]. An alternate approach is collaborative filtering which recommends recipes to similar users. This method escapes the pitfall of recommending repetitive recipes to users and allows variety at the added costs of learning user profiles, which tend to be dually learned with content profiles using approximate matrix factorization methods or through regression on a neural network. Approximate matrix factorization with alternating least squares tends to converge faster than neural network regression but neural network regression allows for more flexibility on how the embeddings are learned. [4.10]

Most modern recommendation systems attempt to extract the benefits of both content-based filtering and collaborative filtering by implementing a hybrid model. For example, Netflix's movie recommendation system uses a combination of collaborative filtering and content-based filtering to maximize users' engagement with the platform. In doing so, it can use content-based filtering to give good initial recommendations while it builds the user profile and recommends newer and diverse films so users do not get bored with the same genre. [4.11]

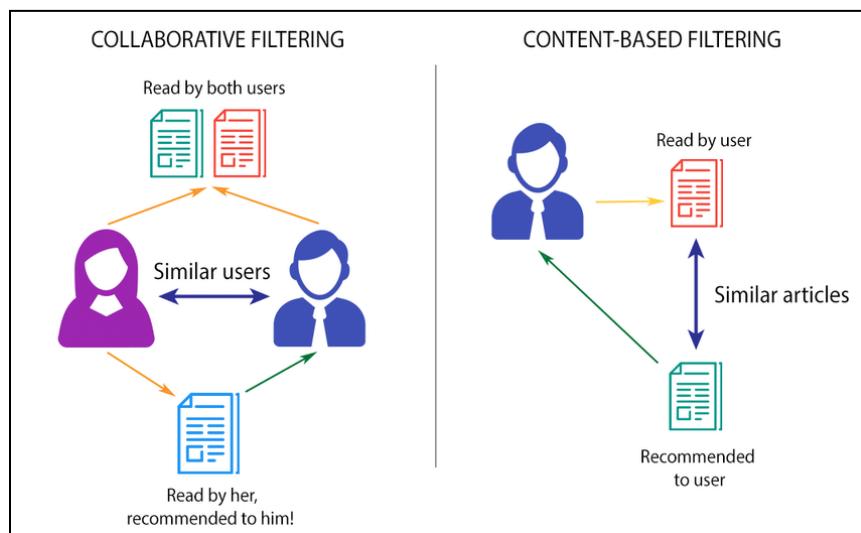


Figure 4.1 (Left) Collaborative Filtering: users with similar preferences are recommended content liked by one. (Right) Content-Based Filtering: similar content is recommended to users based on their historical preferences. [4.3]

One major issue of recommendation systems is the cold start when there has not been a learned profile yet for new recipes or users. In such cases, a neural network-based method may introduce the benefit of alleviating the setbacks from the cold start. Neural networks can be trained on feature data to extrapolate to new data which works in stark contrast to matrix factorization methods which require some user history for effectiveness.

4.4 Recipe Generation System

A reach goal of RecipeCart is to implement a human-aligned recipe generation system. There have been works on attempting this goal some have leveraged knowledge graphs while others have trained pure generative models. A notable issue with recipe generation is that the AI system does not receive direct feedback about the recipes it generates and has no sense of what is intrinsically desired by humans. One way to work around it is to perform a selection process on generated recipes based on user feedback. However, this would not resolve the issue of producing better recipes. Instead, it would prevent the spread of bad ones. An ideal recipe generation model should be able to detect user preferences and tailor some recipes dedicated to users.

In theory, recipe generation should be closely coupled with recipe recommendation. If possible, recipe recommendation should allow a method of modifying recipes based on user input. To that end, a good design choice is to store food in a taste space where certain regions have higher weights than others.

4.4.1 RecipeGPT

RecipeGPT is a method that generates a new recipe using text generative pretraining. It generates new recipes on a token to token basis and is a fine tuned version of GPT-2. It performs pretty well on the data it is given. Given that this is a language model. Using reinforcement learning from human feedback, we can fine tune this model to better respond to user demands and preferences. [4.12]

4.4.2 RecipeMC

RecipeMC builds on RecipeGPT by incorporating a Monte-Carlo Tree Search to maximize the performance of the generated outputs of RecipeGPT. This allows the same functionalities as RecipeGPT, except in exchange for the additional compute, there will be more precision on what is generated and generated responses will be more in line with what has previously succeeded due to the added planning aspect. [4.13]

4.5 Databases

Recipecart's operation depends on three databases and systems: one that processes the input (identifying the ingredients in inventory either in the form of barcode or object detection), one that saves and communicate personalized user's information (login credentials and personal recipes created from the model), and a system that the model uses to take the input and perform reverse embedding search to obtain recipes.

4.5.1 Ingredient Database

In order to effectively turn objects detected from hardware inputs and turn them into food vectors, we look for databases that can effectively look up barcodes and return object information, which can then be filtered into simple names to pass into the main recipe-generating model.

First, we have a look at the UPC Database. The UPC database contains information up to 4.3 million barcode numbers from around the world, which includes the UPC and the EAN numbering systems [4.16]. They also contain an API for barcode lookup with a free starter feature. The other subscriptions are very cheap as well for accessing barcode databases, ranging between \$2.50 and \$25 monthly. The only weakness of the cheap price is the low allowed daily look up rate that reflects on the limits of the subscription. As a starter, it is a good free database to access product barcodes.

Another extensive barcode database is Go-UPC, is aUPC database that contains more than 500 million barcodes for different products and covers across all kinds of industries. The strongest advantage of this database is the amount of niche industries it covers, which includes a lot of liquor brands that may be used, allowing a wider variety of rare recipes that may catch users' interests. One main weakness of this system is the limited number of calls per month in comparison to the price, which starts at \$35 monthly for the cheapest plan in exchange for 5000 API calls [4.17]. In addition, the output is displayed in the form of JSON files (and CSV for more expensive plans) which limits or requires lookup results to go over extra conversion steps.

The last barcode data collection we consider is Upcitemdb. It is similar to Go-UPC, being a massive database with more than 545 million barcodes for different products [4.18]. Upcitemdb contains the advantages of both the listed databases, featuring a limited free option to access through API. Its pricing is more expensive than the other two databases in exchange for more API calls per month.

4.5.2 User Database

In order to personalize users' experience, Recipcart wants to create a database to store signup-information and learned recipe recommenders catered to the users' inventory produced from the learned model. The database systems listed below aim to effectively manage these users' information using common hosting sites such as AWS.

MySQL is one of the most popular databases for user information storage and querying. It is an open-source SQL database management system that is very flexible with dealing with user store and lookups simultaneously. It is simple to pick up, with high performance and scalability by utilizing multi-threading [4.19]. MySQL is free, perfect for smaller scale application developments. One known downside for this system is the low performance towards high-load traffic [4.19]. Overall, MySQL is a good free system to utilize for startup applications with low-moderate user traffic flow, perfect for managing budgets while keeping fast lookups.

MongoDB is a more modern and different approach to the traditional query system. It is a noSQL document-based database that is primarily used for high-load data storage. MongoDB is special, that data is stored in the RAM and also allows indexing, allowing for an all-around quicker access. MongoDB is also easy to set up, and it allows integration of modern JavaScript frameworks. Most importantly, it utilizes horizontal scalability by “sharding” the acquired data, allowing data to be divided and distributed among servers to help optimize overall capacity [4.20]. One weakness of MongoDB is the nature of sharding data makes it hard to duplicate documents, as well as opening the

risk of faulty indexing and resulting in data fault. Similar to mySQL, MongoDB is a good option for budget database systems.

4.5.3 Model Database

The last database Recipescart wants to utilize is a database designed to help with the training process. Here, we look to use lookup vector databases that mirrors the input, then hooked up with the trained model to produce the according trained result. The vector databases listed below help storing vectors to eventually map out an appropriate training plan.

One of the databases under our consideration is Weaviate. It is a free, open-source vector database that can be deployed on the cloud, and piped through ML models to optimize and design specific next-gen search based on our goals. This database matches Recipescart's endgoal, as it is great for recommendation applications by utilizing the trained search mechanics. It is scalable and flexible, allowing for agile project designs. Its' main constraint is the unpopular database does not have much popularity due to its recent deployment and lack of features.

Another database of interest is Pinecone, a vector database that allows for reverse image search. Leveraging reverse image search can allow for classification by approximate nearest neighbor to similar images. This way, by having a few different images of the same object, we can few shot classify different items by comparing their visual similarity to other items. Pinecone allows for an efficient and enterprise-level search of items with the added bonus of enabling efficient search on top of metadata filtering to match certain conditions. However, it is proprietary and will induce a cost in production for a license.

4.6 Backend Hosting Platforms

The RecipeCart backend server is expected to run potentially intensive workloads, ranging from continuous data processing to ML inferencing. To generate and forward results in a timely manner, the backend server needs to be adequately equipped with the proper computing resources. Procuring the necessary hardware to build and host our own server would be too time-consuming and expensive. The alternate—and more ideal—solution would be to deploy and host our backend on a Platform as a Service (PaaS) provider. With a cloud-based server, it would be significantly easier to scale and manage the RecipeCart app based on user traffic and data processing complexity. The added benefits include easier integration with other cloud-based development services, a pay-as-you-go model, high availability and reliability, and higher operational efficiency. We thus narrow our search for a mobile app hosting platform to cloud-based services.

4.6.1 DigitalOcean

DigitalOcean provides Infrastructures as a Service (IaaS) in the form of computing resources and storage space. It offers a relatively simpler and more user-friendly interface and process for app deployment and customization among all the considered cloud computing services [4.21]. As an open-source service, DigitalOcean has extensive

documentation and community support, allowing for much flexibility in app development and integration. The API is well documented and fully comprehensive, allowing for seamless integration with other development tools. It offers scalable and fully managed database options, such that developers can change their storage capacity or redundancy without much server downtime.

DigitalOcean's App Platform allows developers to build and deploy applications through a simplified web interface without direct concern for the underlying infrastructure. App Platform links to a specified GitHub repository or container registry from which it fetches all the backend code and variables [4.22][4.23]. Databases can be simply added to the architecture through a user-friendly configuration panel. DigitalOcean also provides a public URL for the app and can automatically re-deploy it when changes are detected in the backend.

App Platform does not host the mobile app but rather just the backend API and runtime environment. As such, using DigitalOcean for the RecipeCart mobile app would imply incorporating other tools and services into the stack to complete the development and deployment of the mobile app itself [4.21]. DigitalOcean itself is fairly limited in its services and features and thus relies on external tools and add-ons to provide a complete solution.

4.6.2 Firebase

Google's Firebase is a Backend as a Service (BaaS), an abstraction of a PaaS, and features real-time databases, scalable cloud-based hosting, integrated data analytic services, authentication, and various business-tailored machine learning models. Its databases are NoSQL-based and can be managed and updated at millisecond-level speeds. Firebase Authentication maintains UI libraries and SDKs to allow for easy user authentication configurations, leaving little operational overhead for the developers [4.24]. The server hosting and deployment process on Firebase can be simplified to a few clicks and commands on their command center interface. With Firebase, developers can spend less time on creating and managing the backend of their mobile application and focus their attention on enhancing their frontend mobile interface.

Firebase Cloud Functions allow for quick, serverless application deployment. In addition to lessening management efforts, a serverless architecture would put less strain on the backend, by processing user requests and data as they are received [4.25]. Cloud functions are only charged when triggered and are billed based on compute time and frequency [4.26]. Cloud functions can be configured to be triggered based on various events, including alerts and messages from other Google Cloud services.

The primary highlight of incorporating Firebase into the RecipeCart software architecture is Firebase's ML kit, which includes pre-made models for text extraction, face detection, image classification, and barcode scanning among others [4.25].

Using Firebase for the RecipeCart would lead to less development time spent on establishing the backend, allowing for more focus on integration with the hardware sensors and the mobile frontend. The downside is that Firebase is not open-source and

provides relatively little support for and compatibility with other external software technologies. In regards to machine learning, Firebase has no available means for building and deploying custom-made models, forcing developers to primarily rely on the native Firebase ML models [4.25]. Databases are strictly NoSQL and frequent queries can be slow and expensive. Their most relevant features—the Cloud Functions and the ML kit—are not available in their free tier plan.

4.6.3 AWS Amplify

Amazon Web Services's Amplify is a complete—potentially serverless—solution to the mobile development process, featuring tools and services to create the backend and the frontend and provide mobile website hosting. AWS Amplify provides developers with datastores, allowing for easy leveraging of shared and unshared data across multi-platform local persistent storage [4.27]. In addition to the pre-trained ML services, highly sophisticated, custom machine learning models can be built, trained, and deployed through Amazon Sagemaker, AWS's native ML platform, and easily integrated with AWS Amplify. User authentication and management is also facilitated through AWS Amplify's visual interface.

The cross-platform backend environment can be initialized with authentication, GraphQL and REST APIs, and storage in the matter of minutes. Existing backends can be connected through Amplify Libraries and dedicated SDKs [4.28]. A serverless architecture can be adopted by integrating AWS Amplify with AWS Lambda functions—AWS's equivalent of Firebase's Cloud functions. Like Firebase, fully managed database services are offered to significantly minimize the operational overhead. Existing AWS resources can be imported and accessed directly through Amplify, allowing for seamless integration between the mobile app and the AWS cloud.

AWS Amplify Studio allows for accelerated mobile UI development through direct design-to-code integration with Figma [4.27]. Amplify UI Components is an open-source UI toolkit made to easily integrate cloud-based workflows with cross-compatible UI frameworks such as JavaScript, Swift, Flutter, and Android.

Employing AWS Amplify for the RecipeCart would be a learning curve because of the sheer magnitude and complexity of AWS tools and resources. AWS is by far the most complex cloud-based solution among all the considered development platforms.

4.6.4 Azure Mobile Apps

Azure Mobile Apps is a highly scalable and resilient mobile application development platform that provides a framework for authentication, data query, and offline data synchronization. Designed to integrate with Azure App Service, Azure Mobile Apps comes with various community-supported SDKs, enabling cross-platform deployment and facilitated integration with existing enterprise systems [4.29]. Azure Functions, Azure's version of Firebase Cloud Functions and AWS Lambda, allows for serverless compute capabilities and are ideal for implementing the API in a mobile application

backend [4.30]. Like AWS Lambda, the Azure Functions scale based on user traffic and demand and are only billed when invoked.

Custom ML models can be deployed through the Azure Machine Learning workspace and can be subsequently connected to the Azure Mobile backend. The Azure mobile app development process is well documented and community supported—but still less popular than AWS Amplify. Most NoSQL and relational databases are supported on Azure, with several services offering fully managed and scalable database solutions [4.31]. Azure also features the App Center Test, a cloud-based service that allows for automated UI tests on real-world, physical devices.

The drawbacks for using Azure Mobile Apps are in parallel with those of AWS Amplify in terms of complexity, potentially involving a high learning curve. The choice between Azure Mobile Apps and AWS Amplify thus boils down to development preference and familiarity, since architecting and deploying a cloud-based solution is its own complete task. Both services have relatively equivalent pricing options, although Azure is slightly more expensive for general purposes but conversely more cost-efficient for computing purposes [4.32].

| Name | DigitalOcean | Firebase | AWS Amplify | Azure Mobile |
|----------------------------------|-----------------------------------|---------------------------|---------------------------|---------------------------|
| Open-Source Compatibility | Yes | Not natively available | Yes | Yes |
| Custom ML Models | Yes | Not natively available | Yes | Yes |
| Serverless Options | None | Included | Included | Included |
| Mobile Development | Must rely on external SDKs | SDKs included | SDKs included | SDKs included |
| Supported Databases | MongoDB, Kafka, PostgreSQL, MySQL | NoSQL only | Most databases | Most databases |
| Mobile Testing | None | Included | Included | Included |
| Complexity | Low | Medium | High | High |
| Pricing | Starting at \$ 7 / month | Starting at \$ 12 / month | Starting at \$ 14 / month | Starting at \$ 18 / month |

Table 4.3. Backend hosting platform comparisons

4.7 Mobile Frontend Development Platforms

In order to give the customer utmost convenience in optimizing their inventory, RecipeCart aims to launch the frontend platform as an IOS application, then expand to Android to accommodate all devices, and eventually PC web-app. The goal for the frontend is a personalized application with a user-base system that saves and broadcasts the inventory detected from the hardware, as well as the generated recipes from the recipe generation system.

4.7.1 Xamarin

Xamarin is a framework for developing cross-platform mobile applications with .NET (C#). It has built-in, unified support for both Android and IOS, which allows for an easy integration process. A C# native app can be wrapped so that its backend code can be shared with other platforms using a portable or .NET library [4.33], which saves a lot of time for the development process—developing a UI for each different operating system would be tedious and impractical. The framework is backed by Microsoft and is updated frequently.

The primary weakness of Xamarin is that the app building and compiling time is slow, so it is not so useful when trying to emulate more complex apps [4.33]. It is also limited from building more high-end UI or scaling products due to the limited features sacrificed for cross-platforms development.

Overall, Xamarin should be considered when developing a multi-platform application, which is what RecipeCart is aiming for, considering its user-driven goals. Although the framework constrains the developer to a simpler UI, it is enough to develop a simple user-based system, with a basic and easy to look interface that shows them what they have and what they can make.

4.7.2 Flutter

Flutter is a development kit created by Google, designed to develop cross-platform applications. Similar to Xamarin, Flutter allows us to develop applications across different operating systems by rendering a single codebase. A Flutter system contains a framework in Dart, a graphics/accessibility engine in C/C++, and a platform-specific embedder [4.34]. Flutter is simple, as its widgets are easy to customize, and Dart is a relatively easy language to learn. It is also easy to build scalable apps using this framework, as the requirements for Flutter applications are low.

One main weakness of Flutter is the language that it uses. Dart, being a relatively new language, is not a groundbreaking one [4.34]. In comparison to other languages, third party support is not the best, as well as the lack of community support and company recommendation. As a result of this, Dart still falls short compared to other mobile languages such as Kotlin.

Despite Flutter's weaknesses as a relatively new kit with little community and documentational support, application development using this framework has been growing rapidly. It is a strong contender for developing simple mobile applications.

4.7.3 React Native

React Native is an open-source UI framework developed by Meta (Facebook), used to build mobile apps for IOS and Android, as well as compatibility with web browsers. It contains basic core components for the Native framework, and components containing wrappers commonly used for IOS and Android development. NodeJS is used as the foundation to connect the backend with the React application UI [4.35]. React Native applications can be integrated directly with the native code, which allows users to be able to take full advantage of the device [4.36]. It also has a live reload feature that allows the user to quickly view updates, test in the market and gather feedback before smoothly committing to the extensive integration and development processes.

One main weakness of React Native is that due to it running through the JS engine rather than compiling straight to the native code, its performance is lower compared to other frameworks such as Flutter or Xamarin [4.35]. In addition, the nature of layering JS on top of the native code makes updating the React engine or debugging complex application codes more complicated than usual.

Aside from the complications of maintaining React Native applications, its simplicity and ease of integration between different platforms makes it one of the most popular—and still growing—frameworks in the community. The growing public interest and usage also opens the door for support and new developments to libraries and tools. It is a perfect tool for simple projects with a common native code and easy-to-migrate UIs.

| Name | Xamarin | Flutter | React Native |
|----------------------|--------------------------|--------------------------|------------------------------|
| Programming Language | C# / .NET Framework | Dart | JavaScript |
| App Performance | Good, uses native engine | Good, uses native engine | Fair, uses JavaScript engine |
| Community Support | Managed by Microsoft | Open-source / Extensive | Open-source / Extensive |
| UI components | Native | Native | Third Party |

Table 4.4. Mobile frontend framework comparisons

5.0 Standards and Design Constraints

In light of the complexity of the RecipeCart, we review a selection of relevant design standards as well as realistic—or industry-grade—constraints that would help guide our implementation of the main hardware and software frameworks.

5.1 Related Standards

With effective and uniform usage, standards are set as conformance measures to increase efficiency, open trades, encourage widespread consumer confidence and understanding, and potentially reduce costs and efforts relating to accommodating various modalities. Although most components considered for the RecipeCart likely already follow these standards, acknowledging these standards allows for more seamless implementation and integration with other standardized inventions. Most of the relevant standards considered are taken from the Institute of Electrical and Electronics Engineers (IEEE), the International Organization for Standardization (ISO), and the International Electrotechnical Commission (IEC).

5.1.1 Video Packaging and Streaming Standards

Given that the RecipeCart’s main source of data input is video-based, it is imperative to consider standards relating to video packaging and streaming and the appropriate media data format for computer vision applications. A standardized method of compressing and streaming video-based data would allow for efficient and rapid transfer of information across various electronic devices, minimizing the time spent on interpreting and converting video files. Likewise, standardized video formats would promote rapid cross-device communication and ease of information dissipation.

The IEEE 1857 provides efficient coding tool sets for the compression, decompression, and packaging of video input data for telecommunication purposes . The standard defines a set of methods for video encoding and a corresponding decoding procedure [5.1]. The highlighted methods include directional intra-prediction, variable block-size inter-prediction, and context adaptive binary arithmetic coding. The standard targets services involving Internet-based videos, video surveillance footage, IP-based video conference recordings, wide-area television broadcasting, individual user-generated multimedia content, and digital video storage and communication. A popular and industry-grade single board computer—like the Jetson Nano—would typically readily come with the properly standardized video packaging and decompression software such that no additional effort is needed on the developer’s part.

The Open Network Video Interface Forum (ONVIF) provides a common, standard interface between different IP-based physical security devices, regardless of their manufacturing origins [5.2]. The standard’s purpose is to facilitate interoperability and data exchange between differently branded devices. The Real Time Streaming Protocol (RTSP) defines the method for video and audio transmission between two endpoints such

that the latency over an internet connection is minimized. The APIs and libraries supplied by integrated camera manufacturers all follow RTSP-based methods.

The IEEE 2671 standard provides baseline requirements for data format, data quality, application scenarios, and related performance metrics for evaluating and improving online detection deployment. The standard is particularly tailored to address online defect detection through machine vision technologies for manufacturing lines, PLCs, and other backend product lines [5.3]. This standard outlines several approaches and processing pipelines for a detection algorithm software that may help guide the implementation of the RecipeCart computer vision pipeline.

5.1.2 IEEE Federated and Shared Machine Learning Standards

The crux of the RecipeCart lies in its ability to aggregate different machine learning algorithms from various repositories and sources spanned across the Internet. Furthermore, these machine learning models would need to fetch and “crawl” data from various sources owned by distinct entities. The complexity of such an architecture warrants a defined and standardized framework for a federated machine learning system to avoid encountering any privacy or security issues.

The IEEE 3652 standard describes a machine learning framework that allows a model to crawl data distributed across various repositories or devices. The standard blueprints methods of fetching the data while abiding by the appropriate privacy, security, and regulatory requirements. Raw data should not be exchanged directly in a manner that allows a particular party to interfere or interact with the private information of another party [5.4]. With a FML framework, all forms of raw data owned by the various stakeholders are protected by secure, privacy-preserving techniques, such that no information can be leaked or reverse-engineered [5.5].

FML can be categorized into horizontal FML, vertical FML, and federated transfer learning. Horizontal and vertical federated machine learning techniques dictate a method to aggregate data sources that complement each other without directly manipulating the data itself [5.4][5.5]. Horizontal FML specifically concerns datasets with significant overlap in the feature spaces and little overlap in the sample spaces—analogous to adding more data of a similar feature set. Meanwhile, vertical FML addresses the opposite case where the aggregated data share the same sample spaces but not the feature spaces—tantamount to adding features to existing data. Alternatively, datasets that have little overlap in the informational sample space can benefit from FTL which uses transfer learning techniques to exploit reusable and common knowledge shared across different feature spaces.

A federated machine learning architecture should also be sectionalized into layers such that each layer only communicates with the layer directly adjacent to it to minimize unnecessary data exposure and privilege escalations. The IEEE 3652 splits any implementation of FML into five general layers: service, algorithm, operator, infrastructure, and cross-layer [5.4]. The service layer implements the business logic and provides a user service interface to allow FML users to access FML services for modeling and inferencing purposes. The algorithm layer contains the FML algorithmic

logics and directly supports the service layer. The operator layer provides the basic operations necessary for the implementation of the more complex algorithms and includes basic machine learning tools such as aggregators, activations, regularizations, and optimizers. The infrastructure layer provides the raw capabilities of computation, storage, and communication necessary for the operator layer. The cross-layer mainly includes programs and functions for service cataloging, auditing, and monitoring across the other four main layers.

Federated machine learning is especially necessary for IoT-based, edge-computing applications, considering the need to aggregate and analyze data from various platforms to a centralized, cloud-based processing domain. The general FML framework allows independent machine learning users to jointly train a model by supplying their individual and proprietary data under the coordination of a central server [5.4]. FML gives the individual ML user more control over the devices and the accessible data, reducing privacy leakages and communication costs from a centralized perspective.

The IEEE 2830 standard additionally defines a framework for which a machine learning model is trained with encrypted data aggregated from a variety of sources and processed by a third-party trusted execution environment (TEE) [5.6]. The standard discusses shared machine learning as an alternative to FML. In TEE-based SML, the data is encrypted and forwarded to a third-party TEE to construct the model which then becomes the sharing vector. Rather than sharing the data directly, a TEE-based SML opts to compile the data and train the model directly and only share the final deployable model.

The TEE-based SML architecture features a centralized TEE equipped with the appropriate decryption and authentication modules to readily receive the encrypted data from multiple selected sources. The aggregated data is then processed and fed to the ML model as usual and the finalized model is packaged and encrypted for sharing with clients [5.6]. For increased security and privacy, the TEE can combine the crawled data like in FML via horizontal or vertical zipping or transfer learning techniques.

Using a cloud-based machine learning development platform would generally imply implementing a FML or SML framework. The principal ML involvement in the RecipeCart comes from external and readily deployable models that generally would risk violating privacy and security concerns, unless they are regularly maintained by a highly regarded service provider. As such, in compliance with the many privacy and security regulations, the RecipeCart will primarily function on reputable, cloud-based ML platforms.

5.1.3 Network Communication Integration Standards

The RecipeCart communicates with the backend server via wireless communication through a Wi-Fi module that is either readily integrated into the SBC or must be attached externally. With network communication being the primary means of binding all components of the RecipeCart together, this section investigates the standards that guide their implementation in modern communication technology, particularly highlighting the ISO's OSI model and the IEEE 802.11.

The International Organization for Standardization's (ISO) Open Systems Integration (OSI) model dissects the general network communication architecture into seven layers such that standalone systems can communicate on designated layers of operation. Each layer can directly communicate with the layers adjacent to it, but data encoded in a certain layer is encrypted and remains unmodified by subsequent enveloping layers.

Layers are sectioned by functionality such that their respective operations are performed independent and in isolation of one another [5.7]. The physical layer includes the physical mediums and the technologies that transmit data across physical channels such as fiber-optic cables, copper-insulated wires, and air. The data link layer refers to the technologies and endpoints that operate on top of the physical layers such as MAC addresses and Ethernet ports. The network layer lies on top of the data link layer and includes protocols for routing and forwarding packets across different network nodes. The transport layer packages content and ensures that data is transmitted unaltered and in order; it includes protocols such as User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). The session layer maintains user sessions across multiple applications such that streamed data is mapped to the appropriate sessions. The presentation layer refers to the syntax and formatting of the application data such as JSON or HTML. Finally, the application layer consists of user-facing applications and communication protocols such as HTTPS and IMAP.

Generally, a technology would implement a group of adjacent layers rather than the entirety of the OSI model. An outbound transmitted packet would traverse from the application layer to the physical layer, with each layer adding a header to the packet. Conversely, an inbound packet would traverse from the physical layer up to the application layer, with each layer stripping away and interpreting their appropriate header.

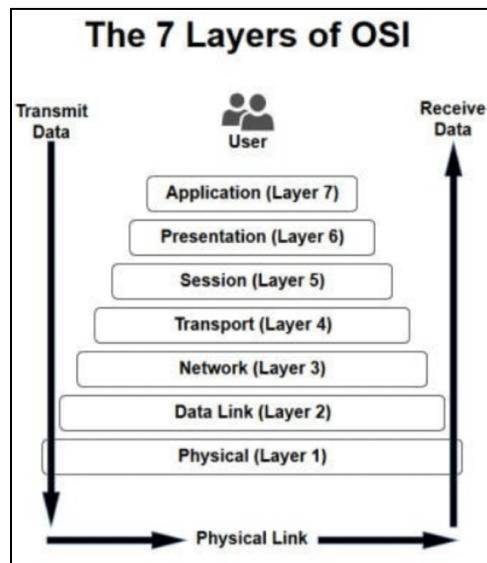


Figure 5.1. The OSI model

The IEEE 802 family of standards consists of specifications for local area networks (LANs), personal area networks (PANs), and metropolitan area networks (MANs). The 802 family contains 24 standards, most of which have been disbanded over the years due to lack of popularity [5.8]. Currently active standards include the 802.1 for higher layer LAN protocols, the 802.3 for Ethernet protocols, the 802.11 for wireless LANs, the 802.15 for wireless PANs, and the 802.18 for radio-based system regulations. These standards are regularly amended to reflect state-of-the-art capabilities and advancements, ultimately shaping the modern mediums of network connectivity.

The IEEE 802.11 was established and maintained since 1997 to specify media access control (MAC) protocols and physical hardware configurations regarding wireless local area network (WLAN) communication [5.9]. The standard describes the services and equipment needed to operate within infrastructure networks as well as in the mobile transitions between such networks. MAC protocols and translation techniques are defined such that wireless communication layers are independent of the device physical origin and do not infringe on user confidentiality. The standard also includes specifications for various common frequency bands such as the 2.4 GHz, the 5 GHz, and the 6 GHz.

Most, if not all, wireless communication protocols and devices follow the 802.11 standard, including but not limited to smartphones, laptops, air printers, and common household IoT gadgets. Wireless communication on any industry-grade SBC complies to the IEEE 802.11 protocols.

5.1.4 Image Sensor Integration Standards

The MIPI CSI-2 protocol is widely adopted for high-speed serial transmission of still and video images from camera sensors to software applications. It allows for a broad range of use cases and applications such as machine vision, imaging, biometric recognition, and contextual awareness [5.10]. The MIPI CCS streamlines the configuration of image sensors on mobile devices by defining a complete command set of functions for implementing and controlling image sensors. This command set allows for rapid integration of the MIPI CSI-2 camera with other devices without needing device-specific drivers [5.11]. The MIPI CSE refers to a set of extensions on the MIPI CSI-2 protocol for automotive and IoT applications. Its inclusion of functional safety enablers and extended virtual channels and data types allow for transmission of higher quality image data, enabling use cases such as robotics and automated visual quality control, autonomous driving systems, and virtual side mirrors [5.12].

5.1.5 Weighing Scale Integration Standards

The IEEE 11073-10415 standard defines the methods of communication between personal weighing scale devices and compute engines, particularly in the context of enabling rapid integration and plug-and-play interoperability. The standard specifies term codes, formats, and behaviors in the context of broadening communication functionalities of weighing devices [5.13]. The standard establishes four general groups of object access services: GET, SET, event report, and action. Concerning the RecipeCart, the GET service is of utmost importance, since it defines the methods for retrieving valuable data

and attributes from the weighing scale. The event report service may be helpful for acknowledging event triggers to indicate result readiness such as when an object has been detected on the scaling platform. The action service meanwhile can be used to switch the scale’s measurement unit between the metric and imperial systems. Alas, integrating the weighing scale should—at the bare minimum—involve maintaining the GET and event report services, and optionally the action service to ensure complete interoperability.

5.1.6 Power Efficient Systems Standards

To minimize the RecipeCart’s power consumption, the proposed system should limit operating in an idle state. The system should perform on low-power mode for the majority of the time, only switching to a more power-demanding mode when necessary. The triggers for dynamically alternating between different power modes may include the visual detection of an object or a direct request from the mobile app. This section highlights some standards regarding energy-efficient architectures, contrasting the advantages between power proportional and low power systems.

The IEEE 1924 standard includes guidelines for the development of power-proportional digital architectures. A power-proportional system would entail energy is consumed only during computations and is reduced during idle, non-operating states [5.14]. In such a system, power supplies must be able to respond to nanosecond variations in the computing load with respect to the complementary metal-oxide semiconductor (CMOS) static power dissipation.

The IEEE 61523 standard defines a format for low power electronic systems, specifically regarding the supply network, switches, power isolation, and power retention. In particular, the standard highlights the Unified Power Format (UPF) which specifies a set of hardware description language (HDL) packages to facilitate the expression of power units and functionalities in digital systems [5.15].

A power proportional system seeks to draw power proportional to the number of computational units or operations, while low power systems are designed to simply reduce the overall power usage in both the on and off states. As such, power proportional systems can scale their power consumption to match computational needs while low power systems cannot. Since the RecipeCart’s workloads are primarily focused on machine learning inferences, we opt for an SBC that supports a power proportional system.

5.1.7 Software Development Standards and Best Practices

Software development standards guide the software systems design and implementation process such that the finished product is consistent, reliable, portable, and scalable. A standardized software also allows for easier management, debugging and testing. This section examines some software development and testing standards as well as common best practices.

The ISO 12207 standard establishes a software development lifecycle framework that is often used to support an agile workspace. The framework itself can be subdivided into primary lifecycle processes, supporting processes, and organizational processes. In complying with the ISO 12207, the project criticality and procedures must be identified and consistently surveyed by a designated administrator. Furthermore, programmers should not be burdened with administrative activities and documentation, which in turn implies that the latter be developed and managed independently of the programming team. This delegation of responsibility ensures that the software can last beyond the employment of the involved entities [5.16].

The ISO 29119 discusses best practices for software and system testing such that testing techniques are not tied to a particular development model. The standard requires documentation of test automation in the form of session sheets with test case specifications, a defect report with the relevant acceptance criteria, and a report on test results [5.17][5.18]. The ISO 30130 defines the framework for software testing tools to enable rapid and continuous deployment of high-quality software [5.19]. The standard explores functional and nonfunctional testing as well as different methods of unit and system testing. Specifically, the ISO 30130 focuses on categorizing software test entities and the associated tools, distinguishing each software testing category, and mapping testing tools to capabilities and potential use cases.

For product consistency, reliability, and portability, software must be developed following common best practices. As emphasized by the ISO 29119 and ISO 30130, system testing should be done early in the development cycle to identify issues and resolve bugs before the code increases in complexity. More importantly, testing should be automated to reduce human error while also allowing for more rapid and diversified testing coverage [5.20]. Code should be simplified and sectionalized whenever possible to improve readability and allow for unit testing. In a similar manner, classes, objects, and functions should occupy their own file. Version control should also be maintained at all times to track changes and enable the necessary rollbacks.

A continuous integration and continuous deployment pipeline leverages automation to roll in new software updates and changes without having to re-integrate the entire existing system [5.21]. The continuous integration phase includes automation for building, testing, and merging the software, while the continuous delivery and deployment involve automatically releasing the updated software in the repository and deploying it to production, respectively.

An agile working environment would allow for collaboration and flexibility by delegating specific tasks and roles to designated people. The agile methodology involves decomposing a particular software project into phases in a cycle of continuous planning, execution, and evaluation [5.22]. The agile methodology is centered around constantly adapting the software to meet customer needs. The RecipeCart would particularly benefit from an agile methodology because of its intrinsic focus on user feedback regarding the recipe generation and recommendation systems as well as the ingredient detection.

5.2 Realistic Design Constraints

In order to realize the Recipecart, a few guidelines are put in place, reflecting different physical aspects that the product must meet in order to comfortably go into deployment. The conditions described below lists different aspects our application must follow in order to ensure accomplishment within the allotted time and budget, as well as safety of the product not only for the end user but also for the surrounding environments.

5.2.1 Financial and Time Constraints

Given the goal of the project, most of the costs should cover the implementation of the hardware to mimic the performance of a smart inventory system with the ability to detect object input, then process and output the information regarding that input. Given the software commitments on the development process of RecipeCart, hardware pricing should be kept minimal despite taking the majority of the budget, and should only perform enough to provide inputs and broadcast to the training model. The remaining budget will mostly be the coverage of hosting database servers on the software side. Depending on the audience the deployment plans to target, the servers hosting cost should cover between a hundred and three hundred concurrent users at a time with at most a two second display latency.

The RecipeCart aims to be a software-heavy project. As a result, we seek to minimize time spent on the hardware aspect, as the main purpose is to mimic smart inventory. This process should not last longer than a third of the development process, and the budget prototype should do enough to at least pass the object input in the form of images to pass information for preliminary training. The software process will occur simultaneously and constantly throughout the whole development period. The majority of time will be spent on developing and optimizing algorithms so that we can maximize our performance on the recipe-generation model. Development of the front-end will also be extensive, as we aim to produce a user-friendly environment, so application interaction should be comfortable and easy to navigate. Recipecart is a solid project that will need to be tested on the market and adjusted according to reviews due to the applications' dependency on users feedback. Therefore, flexible approaches such as Agile should be integrated to quickly deploy the fundamentals of our product, then acquire reviews and update the application accordingly.

5.2.2 Environmental, Social, and Political Constraints

During the development process for Recipecart, aspects regarding the hardware development should be addressed. Firstly, the inventory that will be created to showcase the software prototype aims to be made of a commercial recyclable plastic container that should already have safety-conditions measured [5.23]. The inventory should not have additional build-ons that may cause chemical changes to the inventory itself, and after the showcase it will either be safely disposed of or contained for reusage. Another consideration is regarding the software aspect, knowing that over-dependent on hosting elaborate servers may cause unnecessary waste and energy consumption, hosting servers should be minimized so that it only fits within the designated range of users in mind

[5.23]. Other hardware parts such as cameras and weight measuring technology can also consider reusing old hardwares to minimize environmental waste. The power system, if not designed to integrate to a wall plug, will be selectively chosen so that it is renewable or utilizes solar power. Lithium batteries will be avoided.

The Recipecart, as a software, will be designed so that it will be available to as many users that have access to smart inventory devices. The primary access to the software will be a user-friendly phone app that keeps track of the inventory, generates and suggests different recipes to allow users to create dishes while maximizing their resources. Overall, Recipecart aims to become a convenient add-on that promotes optimization and effective planning in consumers' daily lives.

The Recipecart will be open-source, promoting others to collaborate and improve the software as a whole. It will only collect the user's inventory as input and purely use that data to generate personalized outputs. Other forms of data collection, such as age or household number, will be in the form of optional surveys and abstractive so that the application can improve without taking in unnecessary information that may cause harm to users' privacy.

The Recipecart keeps up to date with any federal restrictions, as well as safety measures and ethical approach towards data collection. As a brand, it is best for the software to reinforce and protect users' data by preventing any kind of information, especially users login information, from potential malwares or leakage.

5.2.3 Health, Privacy, and Safety Constraints

Recipecart should take in the users' physical and mental health into consideration during its deployment. Recipe generations should prioritize healthy products in front in order to promote a healthy lifestyle [5.24]. The application's interface should also be clean and display all products inside inventory to help the user keep track of their storage, which helps relieve some stress of planning and organizing daily lives. Secured data collection and storage is also important, as it relieves the fear of having data breach, as well as building trust between the customer and the application. As a consumer-based product, it is important for Recipecart to uphold these characteristics as an application to deliver all listed features above efficiently to all users to ensure healthy connection and lifestyles.

Most importantly, in order to ensure our users' health throughout Recipecart's deployment, our trained suggestion model must prioritize both high accuracy and precision. The suggested recipes should match with the macros that the average human needs everyday, and should not under- or overestimate any amount that will be suggested to the user to make into reality. Most importantly, the generated recipe must be reasonable and safe for consumption. The model's macro accuracy should be at a 99.0% accuracy, with at least 95.0% precision, and all of these generated recipes must be safe for consumption. Users' health and safety are extremely important, which is why Recipecart keeps a high accuracy standard when it comes to our suggestion model.

As stated above, Recipecart prioritizes users' security. One approach to this is only collecting necessary data and minimizing the extent of data collection. This can be done

by implementing extensive, clear and informed consent processes for data collection and usage. We should also ensure robust measures to protect user data from unauthorized access or breaches [5.24]. The application also aims to implement methods that allow anonymous users behaviors, only allowing learned data to pass when they launch the application and keep it saved in the hardware in case servers get breached beyond our reach. Careful implementation of cloud security structure is also necessary to ensure the promised secured delivery of the product. This is accomplished by complying with relevant safety standards and certifications for hardware or software products, and providing users with necessary instructions and safety guidelines to minimize accidents or misuse. Most importantly, the application should be regularly updated and patched so that the software can address vulnerabilities that could compromise user safety.

5.2.4 Manufacturability and Sustainability Constraints

Recipecart aims to be an add-on software product to smart inventory devices. This means that it can be integrated to different hardware, assuming they have the same input hardware features, and the input information can be piped into the system to run training models. The product should also be scalable, avoiding unconventional methods that do not effectively optimize the overall production stage. It should also avoid environmentally damaging building blocks such as lithium-powered batteries. For the software aspect, the RecipeCart application should be accessed for most devices across different operating systems (iOS and Android for primary phone applications, and potentially expand to Windows and MacOS to cover both platforms) in order to ensure user friendliness.

Regarding sustainability, the Recipecart is more concerned with the longevity of the application. This calls for usage of promising or popular technologies that can guarantee operation for at least 5-7 years, as well as usage of stable libraries that are constantly kept up to date. Older hardware reuse may be considered as a part of the development process in order to minimize waste in general.

6.0 ChatGPT Applications and Limitations

ChatGPT was designed by the artificial intelligence research lab OpenAI, who have made significant contributions to the generative AI including but not limited to GPT-3 which set new benchmarks for generation-based natural language processing, Codex which can be used as the backbone for GitHub Copilot by enabling autocomplete-style suggestions, and Dalle-3, which generates high-resolution images based on text instructions.

ChatGPT has explosively risen to prominence due to its versatility and ability to perform beyond its initial scope. Because of this, Large Language Models have gained much attention from industry giants and scientists alike. Many organizations have begun to incorporate LLMs into their support chatbots and as workflow assistants. Among these LLMs, ChatGPT has distinguished itself as a game-changing technology that many speculate to show early sparks of Artificial General Intelligence.

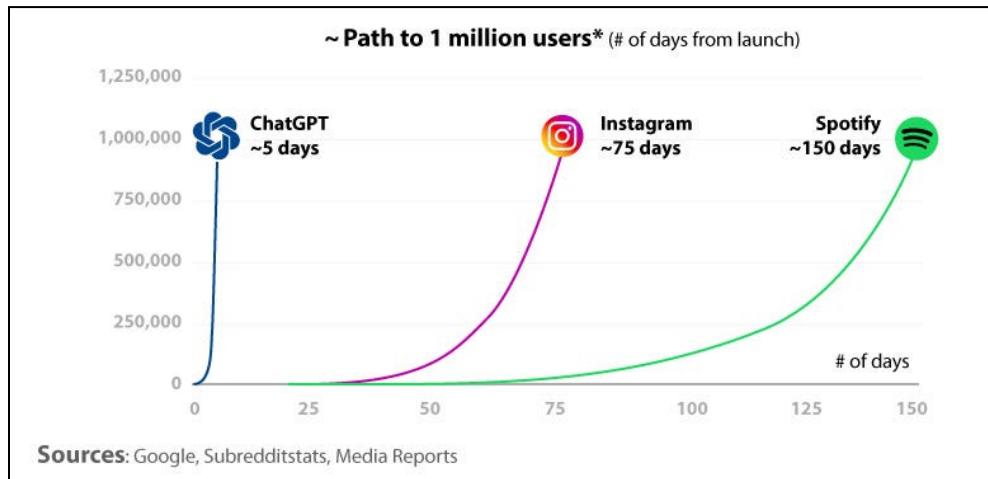


Figure 6.2. Users vs Days since release for ChatGPT vs popular platforms [6.1]

ChatGPT has disrupted much of content creation and search as we know it, with its ability to construct mostly-coherent and human-like responses to user prompts and its extremely user-friendly interface. ChatGPT was trained on a wide range of conversational text and fine-tuned to several specific tasks such as question-answering and dialogue generation.

Here, we discuss ChatGPT and its potential application to our products and its limitations. [6.2]

6.1 Large Language Model Historical Background

Big Tech companies have always dedicated effort towards research on making better and more robust natural language processing models. This focus is driven by LLMs' ability to determine complex dynamics between words and allows companies to tap into the large data reserves of the internet and form meaningful insights on the available data.

Prior to the introduction of the transformer, NLP architecture largely revolved around RNNs and its gated variants like the GRU and LSTM. However, RNNs were subject to instabilities that simple feedforward did not suffer from. Moreover, RNNs due to their design required sequential processing and could not be parallelized, resulting in slow training times and convergence, and most solutions to its stability incurred additional costs to training time. Once the Transformer architecture enabled parallelizable and highly efficient natural language translation. It largely became the core of the success behind current state-of-the-art LLMs. Shortly after its release, Google released BERT which used the encoder of the transformer to receive state-of-the-art in NLP benchmarks. Concurrently, OpenAI released GPT which used the decoder aspect of the transformer to generate human-like text and achieve high accuracy in its performance.

6.2 ChatGPT Architecture

Previous language generation models did not have an interface, making it difficult for non-developers to fully leverage. ChatGPT is designed for non-technical users to be able to use without much configuration. This design philosophy is embedded in the underlying architecture of ChatGPT.

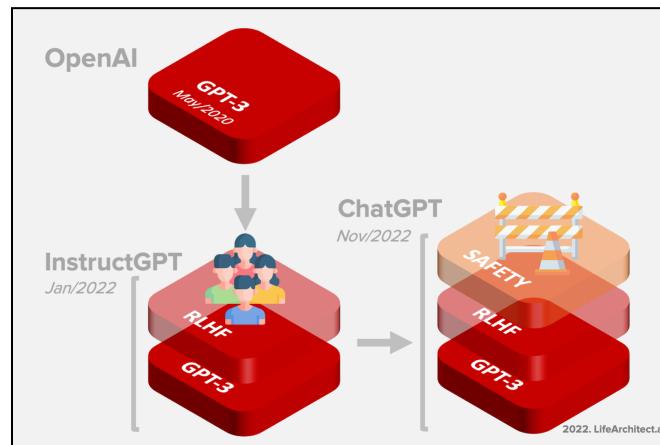


Figure 6.2. ChatGPT Architecture [6.3]. The GPT-3 Model is fine tuned by human-feedback Reinforcement Learning and then a safety layer is added to ensure appropriate information is generated.

6.2.1 Generative Pre-Training

ChatGPT was trained on a pre-trained GPT-3 model. Using the decoder for the Transformer architecture, GPT was trained to predict subwords called tokens in an autoregressive manner—that is to say, one by one. Doing so, GPT was able to learn semantic embeddings for each token, allowing it to emulate context-aware semantically-sound content. [6.3]

6.2.2 Reward Model

Once GPT-3 has been trained, it now needs to produce content that is desired by the user instead of irrelevant and unrestricted human-like blabber. OpenAI's solution to this was to introduce reinforcement learning from human feedback (RLHF). People were requested to write responses to sampled prompts and GPT was fine-tuned to predict desired responses to the prompts. People were then requested to rank generated responses according to the best to the worst. This would train a reinforcement learned reward model which would allow the model to predict which responses best respond to the required context. GPT is then fine-tuned according to the reward model, and this allows for responsive generated content. And this cycle of human feedback and self-supervised reinforcement learning would repeat until the model reached a sufficient level of performance. The model this yielded was called InstructGPT and it was almost ChatGPT except it did not have a filter on what kinds of content it outputted. [6.3]

6.2.3 Content Filter

After GPT has been fine tuned to generate content based on a given task. It must now be filtered to prevent the generation of inaccurate or inappropriate information. This is done by applying a meta-filter over the generated content. Generating appropriate content is framed as a meta prompt to every sub prompt. [6.3]

6.3 Analysis of Originality and Notable Limitations

ChatGPT is hailed for generating "original" content, but there exist some notable caveats to the extent of the originality of the generated content. Given the way the GPT model is trained, where it is trained on regenerating the original content of its dataset, and trained to decrease the difference between its generated content and the dataset. It is severely subject to reiterating content of its dataset without actually generating novel and thought-provoking content. To add to the matter, ChatGPT does not plan its responses nor does it backtrack to correct information. As such, its generated content, more often than not, is some form of reformulated content from the dataset it was given. Contents that seem well-thought out are often more wordy than necessary due to the tokens needed to be placed for the model to process information.

Despite this, ChatGPT is still capable of generating plausible-sounding yet incorrect but original content. In a study, where reviewers were blinded and reviewed abstracts between real content and ChatGPT generated content, 32% of the generated abstracts fooled reviewers into believing it is human-generated content [6.4].

6.4 Prompt Engineering

While ChatGPT is not capable of independent thought, it is a useful tool for eloquuting and verbalizing its given human input. By giving it either sufficient feedback or sufficient content, users can inject knowledge into ChatGPT which can lead to better responses. This insight has led to the rise of prompt-engineering, in which prompts are engineered to draw out more out of the model than it was initially designed to. With detailed and

engineered prompts and given explicit information, ChatGPT can be trained to elicit certain responses that seem to have more coherence. [6.5]

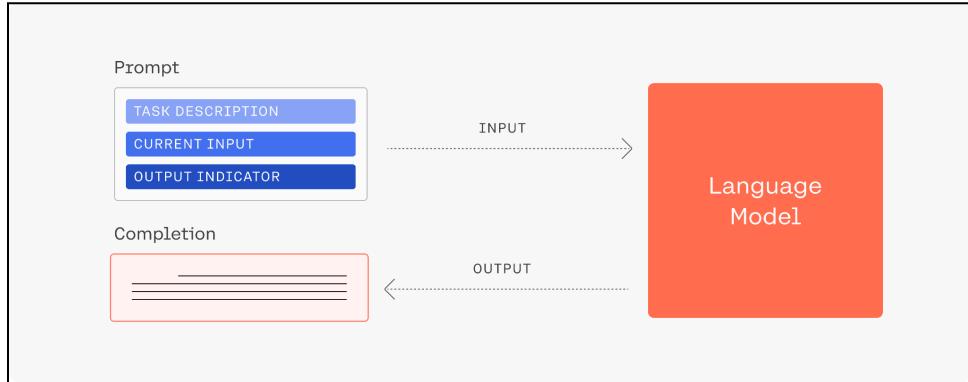


Figure 6.3. Prompt Engineering [6.5]

Users have developed a method to simulate critical thinking in ChatGPT by making it elaborate on its generated content and force it to programmatically perform tasks at a step-by-step basis, resulting in stronger and more consistent performance at logical tasks.

6.5 Potential Integration with RecipeCart

For the purposes of our project, we consider a few options for incorporating LLMs into our model ranging from simply having ChatGPT recommend a recipe to having it react to user feedback and conversationally manage complaints.

6.5.1 Recipe Generation by Prompt Engineering

One option of integrating chat-based generative LLMs into RecipeCart is to prompt the model to generate recipe modification suggestions like increasing the salt or decreasing other ingredients or changing the cooking time. Such changes require a semantic awareness of each cooking instruction and a view of the ingredients involved. By using prompt engineering and user-feedback, slowly but surely the recipe will be fine tuned to the user. However, there are a few noticeable caveats. Unlike in text generation-based training that ChatGPT was able to undergo, recipe-based fine tuning will require the recipe to be cooked and this will cost time and money to make and learn. Additionally, the autoregressive text generation may not be an optimal model for representing recipes and their ingredients.

6.5.2 Review Tagging System

Another option for integrating LLMs is to generate tags for textual reviews that can be aggregated and sent to the recipe author or a recipe generation model for improvement. This is similar to the first idea except that the generation of the recipe is decoupled with the language model, allowing it to use embeddings more appropriate to recipes and ingredients.

6.6 Recent or Related Developments of ChatGPT

Since its initial release in 2021, ChatGPT has undergone a few notable changes that have improved its performance. Firstly, its base model was upgraded with GPT-3.5 and GPT-4, the latter only being available to premium users. The GPT-4 model has more than ten times the number of parameters of the original GPT-3 model, can process images and documents, and can integrate easily with external tools like matlab to compensate for its inability to perform calculations. Though these models enhance the abilities of ChatGPT, many of these modifications do not yet resolve the underlying limitations. [6.6]

Microsoft has incorporated a chat LLM based on the Prometheus model which later became ChatGPT to Bing Search. This has led to AI-assisted searches. However, in the early days of Bing Search AI, there have been numerous attempts to prompt inject Bing AI. In response to this, Microsoft has drastically limited the capabilities of Bing AI, applying limited prompt context, and removing non-search related conversation. [6.7]

In response to ChatGPT, Google introduced Bard, which is a chatbot based on the LaMDA family of models which later became PaLM and PaLM-2. Bard has similar functionalities as ChatGPT except it is able to access data past the year 2021. However, due to some complications during its release, Bard has lost much of the traction it could have gotten at the peak of the Chatbot hype. [6.8]

LLaMA is another chatbot AI developed by Meta. Using only 65 million parameters as compared to GPT-3's millions of parameters, it is trained longer on significantly less parameters than ChatGPT, this makes it more viable for lower end hardware. LLaMa-2 has been released by Meta and the model is available on a case-by-case basis. [6.9]

Concurrently, there have been attempts to extend ChatGPT by iteratively prompting it to elaborate on its own outputs to form more and more clear pictures of abstract ideas. One notable model is the Voyager model which uses GPT-4 to code functions through iterative prompting for navigating Minecraft or other exploratory games, and construct a skill library which can be stored and retrieved from a vector database. From this result, the power of the zero-shot generalization that LLMs can provide brightens many avenues for research in extracting knowledge from LLMs. [6.10]

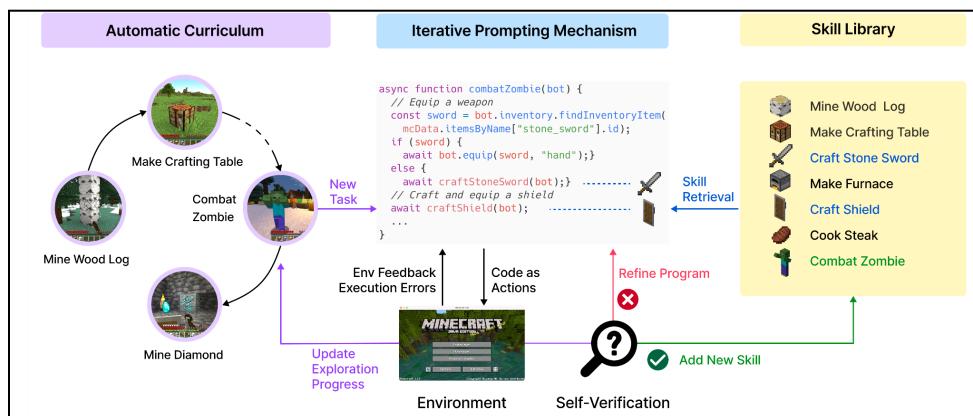


Figure 6.4. Sketch of Voyager's Pipeline. [6.10]

7.0 Hardware Design

fd

7.1 General Structural Outline

fsdfa

7.2 Jetson Nano Schematics Walkthrough

7.3 Camera Placement and Integration

7.4 Weight Scale Platform Placement and Integration

8.0 Software Design

15 pages here

9.0 Implementation and Prototype Workflow

10 pages here

10.0 System Testing

10 pages here

11.0 Administrative Content

11.1 Estimated Budget and Financing

| Parts name | Description | Price |
|--------------------------|---|-------|
| SBC Module | For data pre-processing, basic machine learning computations, and system control | \$150 |
| Wide-Angle Camera Module | 3280x2464 resolution cameras | \$36 |
| Wide-Area Digital Scale | Digital scale that can integrate with PCBs and MCUs | \$40 |
| Wireless NIC module | For connecting the SBC to the internet | \$22 |
| Batteries (x4) | To provide power | \$100 |
| Wires (x100) | Wires for connecting hardware components | \$15 |
| Large Plastic Container | Plastic container wide enough to simulate a tray in the fridge or a shopping cart | \$20 |
| Miscellaneous Items | Tape, glue, etc. | \$20 |
| | GRAND TOTAL: | \$403 |

Table 11.1. Estimated Budget

The proposed budget assumes the higher-end estimated values of the listed parts. Depending on additional needs for barcode and object detection, an extra camera module may be needed. Achieving advanced and stretch goals would also entail a more extended budget. Currently, this project does not have any sponsors; funding will be at the expense of the individual team members.

11.2 Project Timeline and Milestones

| Milestone | Start Date | End Date |
|--|-------------|-------------|
| Brainstorm, pitch ideas, and form groups | 21-Aug-2023 | 27-Aug-2023 |
| Flesh out ideas and do preliminary research | 28-Aug-2023 | 03-Sep-2023 |
| Invite ECE professors and faculty to join the senior design reviewer committee | 01-Sep-2023 | 15-Sep-2023 |
| Draft Chapter 2: Project Description Draft Chapter 11: Administrative Content | 04-Sep-2023 | 15-Sep-2023 |
| Design the website and upload the 10-page D&C | 17-Sep-2023 | 06-Oct-2023 |
| Meet with Dr. Wei and review the 10-page D&C | 19-Sep-2023 | 22-Sep-2023 |
| Draft Chapter 3: Hardware Parts Comparisons | 25-Sep-2023 | 08-Oct-2023 |
| Draft Chapter 4: Software Comparisons | 09-Oct-2023 | 24-Oct-2023 |
| Draft Chapter 5: Standards and Design Constraints Draft Chapter 6: ChatGPT Applications and Limitations | 25-Oct-2023 | 03-Nov-2023 |
| Draft Chapter 7: Hardware Design Draft Chapter 8: Software Design | 04-Nov-2023 | 18-Nov-2023 |
| Finalize and upload the 60-page D&C to the website | 04-Nov-2023 | 05-Nov-2023 |
| Meet with Dr. Wei and review the 60-page D&C | 06-Nov-2023 | 08-Nov-2023 |
| Draft Chapter 9: Implementation and Prototyping | 15-Nov-2023 | 24-Nov-2023 |
| Draft Chapter 10: System Testing | 25-Nov-2023 | 02-Dec-2023 |
| Conclude and submit the Senior Design I Report | 03-Dec-2023 | 05-Dec-2023 |
| END OF SENIOR DESIGN I | | |
| Gather relevant hardware parts, including cameras, SBC, wires, batteries, and suitable container | 12-Dec-2023 | 29-Dec-2023 |
| Acquire permissions for necessary software, including pre-trained ML models, APIs, and database access | 12-Dec-2023 | 29-Dec-2023 |
| Build physical architecture prototype | 08-Jan-2024 | 28-Jan-2024 |
| Integrate relevant ML APIs for ingredient detection | 08-Jan-2024 | 28-Jan-2024 |

| | | |
|---|-------------|-------------|
| Integrate physical architecture with backend software | 28-Jan-2024 | 11-Feb-2024 |
| Develop and integrate recipe recommender system | 05-Feb-2024 | 04-Mar-2024 |
| Develop mobile frontend and connect backend database | 11-Feb-2024 | 11-Mar-2024 |
| Debugging and testing until working and presentable | 11-Mar-2024 | 15-Apr-2024 |
| Finalize report, website, presentation, and demo | 15-Apr-2024 | 28-Apr-2024 |

Table 11.2. Project Milestones

11.3 Project Roles and Work Distribution

12.0 Project Conclusion

2 pages here

Appendix A

References

- Lenovo. (2022). Revolutionizing the shopping experience. *Lenovo.Com*. Lenovo. Retrieved September 15, 2023, from <https://www.lenovo.com/content/dam/lenovo/dcg/global/en/customer-stories/case-study-cust2mate.pdf>.
- Samsung (2023). *Samsung Family HubTM: Samsung Us: Undefined undefined*. Samsung us. (2023, July 29). <https://www.samsung.com/us/explore/family-hub-refrigerator/overview/>.
- Slater, M., & Kreizman, J. (2022). (rep.). *A2Z Smart Technologies Initiation of Coverage*. Valore Research & Consulting. Retrieved September 15, 2023, from https://a2zas.com/wp-content/uploads/2022/06/AZ_Valore_Initiation_April_2022_ENG_DRAFT_compressed.pdf.
- [2.4] <https://samsungfood.com/about/#:~:text=The%20Whisk%20app%20officially%20rebranded,press%2C%20users%2C%20and%20creators>.
- [2.5] <https://app.samsungfood.com/recipes>
- [3.1] <https://www.techtarget.com/iotagenda/definition/microcontroller#:~:text=A%20microcontroller%20is%20a%20compact,peripherals%20on%20a%20single%20chip>.
- [3.2] <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf>
- [3.3] <https://www.seeedstudio.com/blog/2019/10/24/microcontrollers-for-machine-learning-and-ai/>
- [3.4] <https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf>
- [3.5] <https://coral.ai/products/dev-board/#tech-specs>
- [3.6] <https://developer.nvidia.com/embedded/jetson-nano>
- [3.7] <https://forums.developer.nvidia.com/t/best-cameras-for-jetson/49494>
- [3.8] <https://www.raspberrypi.com/documentation/accessories/camera.html#camera-module-3>
- [3.9] <https://boredconsultant.com/2023/02/19/Raspi-Camera-Module-v3-vs-HQ-Camera/>

- [3.10] <https://www.logitech.com/en-us/products/webcams/c920s-pro-hd-webcam.960-001257.html>
- [3.11] <https://drive.google.com/file/d/10IgEGNXSWZNjBNJv240IYPmdfYQsOpE6/view>
- [3.12] <https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide/all#installing-the-hx711-arduino-library-and-examples>
- [3.13] <https://community.particle.io/t/strange-behavior-of-load-cell-hx711/49566/2>
- [3.14] <https://www.digitalscalesblog.com/connecting-scale-raspberry-pi/>
- [3.15] <https://laumas-us.com/product/tlc-load-cell-amplifier/>
- [3.16] <https://www.seeedstudio.com/ZKETECH-EBD-A20H-AC-Electronic-Load-Battery-Capacity-Tester-Power-Supply-Tester-30V-20A-200W-p-4521.html>
- [3.17] <https://www.findthisbest.com/best-lab-power-supplies>
- [3.18] <https://www.digikey.com/en/products/detail/xp-power/VCS50US12/4488662>
- [4.1] <https://www.dynamsoft.com/barcode-reader/docs/core/introduction/>
- [4.2] <https://github.com/nrl-ai/daisykit>
- [4.3] <https://www.themarketingtechnologist.co/building-a-recommendation-engine-for-geeksetting-up-the-prerequisites-13/>
- [4.4] <https://neptune.ai/blog/understanding-few-shot-learning-in-computer-vision>
- [4.5] <https://cloud.google.com/vision?hl=en>
- [4.6] <https://aws.amazon.com/rekognition/>
- [4.7] <https://webuters.medium.com/google-cloud-vision-vs-amazon-rekognition-which-is-better-c7cb4780d82a>
- [4.8] <https://ai.meta.com/blog/dino-v2-computer-vision-self-supervised-learning/>
- [4.9] <https://developers.google.com/machine-learning/recommendation/content-based/basics#:~:text=Content-based%20filtering%20uses%20item.previous%20actions%20or%20explicit%20feedback.>

[4.10] <https://developers.google.com/machine-learning/recommendation/collaborative/basics>

[4.11] <https://www.scientificamerican.com/article/how-recommendation-algorithms-work-and-why-they-may-miss-the-mark/#:~:text=Most%20recommendation%20algorithms%20now%20use,of%20release%20and%20other%20attributes.>

[4.12] <https://recipegpt.art>

[4.13] <https://krntneja.github.io/files/recipe-generation-paper-iccc23.pdf>

[4.16] <https://upcdatabase.org/api-pricing>

[4.17] <https://go-upc.com/plans/bulk-lookups>

[4.18] <https://www.upcitemdb.com/>

[4.19] <https://blueclawdb.com/mysql/advantages-disadvantages-mysql/>

[4.20] <https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb>

[4.21] <https://nestify.io/blog/exploring-digitalocean/>

[4.22] <https://docs.digitalocean.com/products/app-platform/how-to/create-apps/>

[4.23] <https://www.digitalocean.com/products/app-platform>

[4.24] <https://www.geeksforgeeks.org/firebase-introduction/>

[4.25] https://blog.back4app.com/firebase/#Firebase_Advantages

[4.26] <https://firebase.google.com/docs/functions>

[4.27] <https://blog.back4app.com/aws-amplify-vs-lambda/#:~:text=AWS%20Amplify%20allows%20developers%20to,intelligence%20and%20machine%20learning%20actions>

[4.28] <https://aws.amazon.com/amplify/>

[4.29] <https://learn.microsoft.com/en-us/azure/developer/mobile-apps/azure-mobile-apps/overview>

[4.30] <https://learn.microsoft.com/en-us/azure/developer/mobile-apps/serverless-compute>

[4.31] <https://azure.microsoft.com/en-us/products/category/databases>

[4.32]

<https://www.c-metric.com/blog/cloud-pricing-comparison-of-aws-vs-azure-vs-google-cloud/#:~:text=From%20this%20comparison%2C%20it's%20very,rest%20two.>

[4.33]<https://dotnet.microsoft.com/en-us/platform/support/policy/xamarin#:~:text=Xamarin%20support%20will%20end%20on,about%20upgrading%20Xamarin%20projects%20to%20>

[4.34]<https://waverleysoftware.com/blog/why-use-flutter-pros-and-cons/#:~:text=Flutter%20allows%20developers%20to%20build,application%20development%20are%20much%20lower.>

[4.35]<https://reactnative.dev/docs/components-and-apis#android-components-and-apis>

[4.36]<https://pagepro.co/blog/react-native-pros-and-cons/>

[5.1] <https://ieeexplore.ieee.org/document/6783681>

[5.2] <https://www.pelco.com/blog/onvif-guide>

[5.3] <https://ieeexplore.ieee.org/document/10011140>

[5.4] <https://ieeexplore.ieee.org/document/9382202>

[5.5] <https://ieeexplore.ieee.org/document/9456823>

[5.6] <https://ieeexplore.ieee.org/document/9586768>

[5.7]

[https://aws.amazon.com/what-is/osi-model/#:~:text=The%20Open%20Systems%20Interconnection%20\(OSI\)%20model%20was%20developed%20by%20the,IEC%207498%2D1%3A1994.](https://aws.amazon.com/what-is/osi-model/#:~:text=The%20Open%20Systems%20Interconnection%20(OSI)%20model%20was%20developed%20by%20the,IEC%207498%2D1%3A1994.)

[5.8] <https://www.ieee802.org/>

[5.9] <https://ieeexplore.ieee.org/document/6522432>

[5.10] <https://www.mipi.org/specifications/csi-2>

[5.11] <https://www.mipi.org/specifications/camera-command-set>

[5.12] <https://www.mipi.org/specifications/mipi-cse>

[5.13] <https://ieeexplore.ieee.org/document/9082285>

[5.14] <https://ieeexplore.ieee.org/document/9968219>

- [5.15] <https://ieeexplore.ieee.org/document/7076554>
- [5.16] <https://www.iso.org/standard/63712.html>
- [5.17]
<https://www.softkraft.co/software-development-standards/#iso-12207-software-life-cycle-processes>
- [5.18] <https://www.iso.org/standard/79428.html>
- [5.19] <https://www.iso.org/standard/64901.html>
- [5.20]
<https://www.opslevel.com/resources/standards-in-software-development-and-9-best-practices>
- [5.21] <https://www.redhat.com/en/topics/devops/what-is-ci-cd>
- [5.22] <https://asana.com/resources/agile-methodology>,
- [5.23] <https://ieeexplore.ieee.org/document/9062658>
- [5.24] <https://ieeexplore.ieee.org/document/10278119>
- [6.1]
<https://medium.com/@rushdauwaiz/exploration-of-chatgpt-and-the-future-of-generation-ai-1-1767fef6e137>
- [6.2] <https://eliiza.com.au/wp-content/uploads/2023/03/ChatGPT-decoded.pdf>
- [6.3] <https://lifearchitect.ai/chatgpt/>
- [6.4] <https://www.nature.com/articles/s41746-023-00819-6>
- [6.5] <https://www.thepromptmarket.ai/prompt-engineering-info/>
- [6.6] <https://openai.com/gpt-4>
- [6.7]
<https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/>
- [6.8] <https://blog.google/technology/ai/bard-google-ai-search-updates/>
- [6.9] <https://ai.meta.com/blog/large-language-model-llama-meta-ai/>
- [6.10] <https://voyager.minedojo.org>