RecipeCart: A Food Leftovers Solution

Darren Ha
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, United States
da771216@ucf.edu

Quan Nguyen
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, United States
qu424618@ucf.edu

Bryan Ha
Department of Computer Science
University of Central Florida
Orlando, United States
br476638@ucf.edu

***Abstract*—The RecipeCart encompasses a modern solution to consumerism habits, providing users with alternatives to their food leftover dilemmas. Recipes are recommended to users based on the ingredients available to them, which directly reflects the user's physical inventory. Ingredients are dynamically identified through computer vision algorithms using a mini computer, and server-side logistics are handled through cloud-based services. This paper discusses the project's goals and constraints, the principal hardware and software components, the designs and methodologies employed, and system testing procedures and results.**

***Index Terms*—cloud computing, computer vision, IoT, mobile app, NoSQL, recommendation system, vector embedding**

I. Introduction

In this modern consumer-centric economy, a consumer's speculation of their needs often leads to them buying more supplies than necessary, increasing the ratio of unused and wasted resources. Correcting and enforcing new spending behaviors would most effectively address the over-consumption problem but is not realistically achievable in the short term. Rather, the simpler approach would be to accommodate the current spending habits. The RecipeCart attempts to encourage the usage of leftover resources in the kitchen by generating recipe recommendations based on an ingredient inventory.

Many companies, like Samsung and Amazon, have developed smart fridges and smart shopping carts that prove to be effective tools for encouraging consumers to better manage their groceries. Generative AI researchers have meanwhile made significant advancements in creating systems capable of generating novel recipes given text prompts. Various mobile apps have been developed to allow users to explore recipe ideas and manage their food inventory. However, while each technology provides a solution to a specific step along the meal preparation "pipeline," none of them is fully encompassing or end-to-end.

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

This interdisciplinary project was self-sponsored by the team members.

II. Project Goals and Objectives

A. Goals

The primary goal of the RecipeCart is to simply reduce the waste of leftover ingredients by creatively re-purposing them with novel recipe suggestions. The RecipeCart intelligently identifies the ingredients being presented and subsequently produces a list of user-relevant recipes. Recipes are fetched from an internal database, custom-filtered and sorted based on user needs, and displayed on a mobile platform for user interaction.

In designing a seamless input process for the end user, video streaming the food items is strongly considered. Video streaming the input would allow for dynamic interactions with 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 poses an extreme challenge in both complexity and resource, making this feature a stretch goal.

Another advanced goal is to integrate a scaling system into the RecipeCart to quantify the ingredients and suggest recipes that more accurately reflect the state of the leftovers.

Although generating creative and composite recipes through AI has already been done multiple times, adding this feature is a stretch goal because the resulting recipes do not necessarily guarantee a similar level of delicacy. Not only do 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 requires a significantly more advanced generative model and also a system to determine the validity of the recipe.

B. Objectives

In achieving the project goals, an ensemble of pre-trained machine learning models balancing between object classification and barcode detection must be implemented to accurately identify the input ingredient. The object classifying model must classify visually distinct unpackaged groceries based on vector encoding similarities, while the second model must specifically detect for barcodes on packaged products and query an external barcode database for the product metadata. Both models must be concurrently available to provide a flexible ingredient detection process for the users.

To capture pictures of the food item from a central point of view, the camera must be mounted at the front of the container—pointing inwards at an appropriate tilt—and be connected to a single board computer (SBC) for centralized power and control.

The SBC itself must be equipped with the appropriate network adapters for easy communication with the backend server. The appropriate power sources must be attached to the hardware modules to simulate the RecipeCart as an add-on feature to existing technologies such as the Amazon DashCart and Samsung's Smart Fridge.

A server must be developed to host the backend business logic to pre-process the input images, provide a repository for machine learning logistics, and query the external barcode database. The server must also host its own databases to maintain the basic user metadata, recipes, ingredients, and other information necessary to tune the recipe recommendation system.

The mobile user interface must be fronted by a basic authentication system and allow users to dynamically update their recipe preferences and also initiate requests 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.

III. Project Specifications

Engineering constraints are established to provide a guideline in developing the RecipeCart. Given the project's focus on computer vision, the most important design specifications involve achieving a low ingredient classification time and a decently high classification accuracy. Recipe recommendation time is another measure of interest relating to the program's ability to quickly return a shortlist of user-specific recipes.

TABLE I: Engineering Design Specifications

Specifications	Value
Object Classification Time	$<$ 5 seconds
Object Classification Accuracy	$>$ 80 %
Recipe Recommendation Time	$<$ 5 seconds
Min. Number of Recipe Outputs	$>$ 1 recipe
Absolute Weight Error	$<$ 20 grams
Broadcast Delay	$<$ 2 seconds
Power Consumption	15 Watts
Product Weight	$<$ 4 kg
Budget	$<$ $400

IV. Hardware Components

The RecipeCart's principal hardware components include the single board computer (SBC), the camera sensor, the load cell amplifier, the digital scale, the weight sensor, the power supply system, and the container. This section examines the hardware technologies selected for each component in the hardware architecture.

A. Single Board Computer

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 Google's Coral Dev Board, all the while being more powerful than the Raspberry Pi 4. The Jetson Nano supports not only TensorFlow but also PyTorch, Keras, and scikit-learn among other machine learning frameworks. It also has many software developer kits (SDKs) 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 [6]. This SBC comes with 12 lanes of MIPI CSI-2 for camera sensors, several USB connectors, and its own active cooling system.

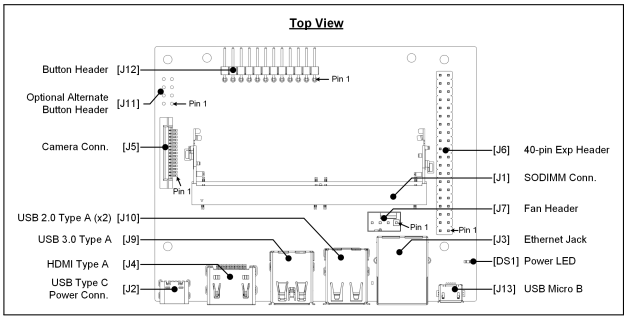

Fig. 1: Jetson Nano Carrier Board Schematic

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.

Nonetheless, the Jetson Nano provides adequate compute power to perform the necessary machine learning workloads on the edge while maintaining compatibility with a variety of machine learning frameworks.

B. Camera Sensor

In regards to camera sensors, we gravitate toward camera serial interface (CSI) modules rather than the traditional USB cameras due to the former's direct integration with the Jetson Nano and customization flexibility. They are generally smaller in form factor and provide low-level control, allowing for hardware performance maximization.

Raspberry Pi camera modules are natively supported by the Jetson Nano. We originally opted for the Raspberry Pi Camera Module 3—publicly released in January 2023—for its 12-megapixel Sony IMX708 image sensor with motorized autofocus, capable of recording full high definition videos at 50 fps with a very high signal-to-noise ratio [8]. However, we soon discovered that the Jetson Nano did not yet have the necessary drivers to operate the Camera Module 3; we hence reverted to the Raspberry Pi Camera Module 2.

The Raspberry Pi Camera Module 2 has an 8-megapixel Sony IMX219 sensor, allowing it to record 720p videos at

60 frames per second [8]. Its focus can be adjusted, albeit manually via a pair of tweezers. By default, its focus is set to 50 cm—from the camera lens—which required us to manually rotate the outer lens apparatus to shorten the focus to 10 cm.

C. Load Cell Amplifier

The HX711 load cell amplifier acts as a bridge between the digital scale and the Jetson Nano. It receives the analog signals from the scale's load cells, amplifies, and subsequently relays them to the Jetson's GPIO pins.

A recurring problem in implementing the HX711 stems from its fluctuating value reports due to temperature affecting the performance load cells as well as the nature of directly interpreting voltage signals. In addition, the module also suffers from value drifting errors due to the circuit being sensitive to supply voltage or other mechanical issues [7].

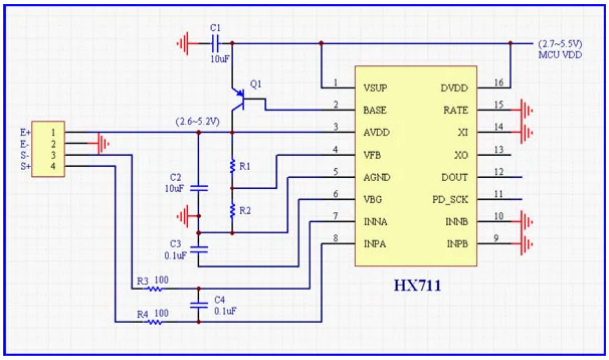

Fig. 2: HX711 Load Cell Amplifier Schematic

D. Digital Scale

In addressing our advanced goal to capture the weight and quantity of certain label-less ingredients—i.e. unpackaged fruits and vegetables—we must select a digital scale of adequate precision. Immediately, we narrowed our search down to kitchen scales for their compactness and increased unit precision over bathroom scales. An additional requirement is that the scale's circuit board must be accessible without completely displacing the scale's backboard and its load cells.

We selected the GGQ digital kitchen scale with its load limit of 15 kg mainly because of the aforementioned accessibility requirement. The circuit board can be easily accessed and rewired to the HX711 without completely exposing the scale's underbelly.

E. Power Supply System

With the RecipeCart serving as an add-on feature to existing technologies such as a smart cart or a smart fridge, the power supply system in question is directly related to its use case. Evidently, mounting the RecipeCart onto a smart shopping cart would necessitate a set of batteries capable of powering the Jetson Nano and its 10 Watts power consumption. Alternatively, integrating the RecipeCart into a smart fridge would transfer the power supply burden to the fridge, which would be directly connected to a power outlet.

For the demonstration of this project, we present the RecipeCart as an additional resource to the smart fridge. Thus, the entire hardware architecture is simply powered by a conventional 5VDC 4A power jack, which is the common wall-plug solution for the Jetson Nano.

F. Plastic Container

The plastic container is the vehicle on which the Jetson Nano and other associated modules are mounted. The selected container must be wide enough to house the digital kitchen scale and deep enough to provide ample space to temporarily hold moderately sized items. An 11.4-quart Rubbermaid dishpan with dimensions 14.5 x 12.5 x 5.6 (L x W x H) is selected for this purpose.

V. Software Components

This section discusses the available software that is implemented as the core features of the RecipeCart. The different software components are selected based on their scalability, efficacy, and cost of service.

A. Barcode Detection Software

Barcode detection is the simplest and most efficient method of identifying ingredients. Since the content of the barcode contains an exact product number, it allows for precise and consistent classification of items simply by querying a universal product code (UPC) database.

To minimize software-related costs, we decided to implement the barcode detection algorithm using the open-sourced PyZbar library. PyZbar employs other open-sourced Python libraries such as PIL, OpenCV, and numpy to dynamically detect and decode barcodes. Since Python is noticeably slower than any compiled implementation in C/C++, the barcode detection software suffers a slightly higher latency. However, this discrepancy is negligible given our flexible classification time specifications.

The barcode detection and decoding script is deployed onto the Jetson Nano while the barcode parsing and querying script is performed on the server.

B. Image Classification Software

The image classification software module is intended to classify items whose barcodes are deformed or simply absent such as fresh unpackaged produce. Training a scalable and robust classification model is essential to minimizing the ingredient detection time while retaining a high classification accuracy. The image classification module is deployed on a server to enable more powerful and scalable computations.

The naïve approach 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 (CNNs) 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 of different products and variations. Furthermore, having a larger number of classes does not scale

well with standard classification networks: the model must be retrained for every new class or ingredient, which is extremely costly in the long-term.

An ideal classification model must be able to learn to identify new ingredients quickly. Thus, we seek a general encoder model that will store latent vector embeddings for different items and use another algorithm to classify the encoded data. This method instead allows us to do a scalable reverse search on items that have been detected.

DINOv2 is Meta's pre-trained computer vision model, capable of detecting depth, segmentation, and instance retrieval at high accuracy with few-shot capabilities. It is also open-sourced and can be implemented and integrated on the developer end [5], allowing for minimal developing costs aside from server maintenance. When combined with a vector database for similarity search, users can perform efficient search and detection of products without having to completely retrain the underlying model.

C. Cloud-Based Backend Platform

The RecipeCart backend server runs potentially intensive workloads, ranging from continuous data processing to ML inferencing. 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.

Amazon Web Services's (AWS) Amplify is a complete and potentially serverless solution to the mobile development process, providing developers with datastores, allowing for easy leveraging of shared and unshared data across multi-platform local persistent storage [1]. Additionally, user authentication and management is also facilitated through Amazon Cognito, AWS's authentication system.

A serverless architecture can be adopted by integrating AWS Amplify with AWS Lambda functions and Amazon DynamoDB, which can significantly minimize the operational overhead. Existing AWS resources can also be imported and accessed directly through Amplify, allowing for seamless integration between the mobile app and the AWS cloud.

D. Databases and Storage

With the backend server deployed on AWS, we are more inclined to build the rest of our processes with AWS services. We store the recipes, ingredients, and user settings data in tables in Amazon's—serverless, NoSQL database—DynamoDB. DynamoDB is chosen over relational databases like MySQL and PostgreSQL for its scalability, allowing it to perform at millisecond-level speed even when dealing with large data. It is also natively integrated with AWS Amplify.

While ingredient barcodes are intended to store locally in DynamoDB, the barcode metadata is initially fetched from World Open Food Facts (WOFF), a non-profit food products database that aggregates food products at the global scale [11]. The UPC information is directly queried from WOFF and parsed on the server-side. As more ingredient products are populated into DynamoDB, the RecipeCart app is expected to rely less on the WOFF API and database.

To implement the object classification pipeline with Meta Dinov2, we seek a vector database that can be deployed on the cloud and piped through ML models to optimize and design specific next-gen search. Weaviate is selected for its open-source background as well as its scalability and flexibility, allowing for agile project designs [10]. Its' main constraint is its recent deployment and lack of features, attributed to its limited community support.

Amazon's Simple Storage Service (S3) acts as the intermediary layer between the RecipeCart mobile app and the backend resources, housing software packages and data needed for machine learning computations.

E. Mobile Frontend Development Framework

The goal for the mobile frontend is a personalized application with a user-base system that saves and displays the inventory detected from the hardware, as well as the recipes from the recipe recommendation system.

The mobile app is developed in Dart, following the Flutter framework. 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.

VI. Hardware Design and Implementation

The hardware design process simply involves the direct integration of the camera and weight scaling system with the SBC, while providing an adequate and reliable power supply to all of them. Having obtained the permissions to use the Jetson Nano as a complete solution to the microcontroller and PCB design requirements, the brunt of the hardware-related workload is dedicated to the implementation of a weighting system with the HX711 load cell amplifier and an external scaling platform. In contrast, the installation and configuration of the Raspberry Pi camera is relatively trivial, given its compatibility with the Jetson Nano.

A. Physical Designs

The purpose of the RecipeCart is to demonstrate the flexibility and compactness of the ingredient detection solution. All the hardware components are thus assembled onto a portable container. With the SBC functioning as the heart of the architecture, the camera and loading platform are placed in close proximity to the Jetson Nano to minimize broadcast delay and wire entanglement.

B. Weight System Design

The HX711 is directly connected to the circuit board embedded inside the kitchen scale's main circuitry compartment. The E+ pin refers to the positive end of the board and connects to its V_{cc}, while the E- pin refers to the negative end or its ground. The A+ and A- pins represent the amplifier circuits and are connected to the white and green/blue wires on the

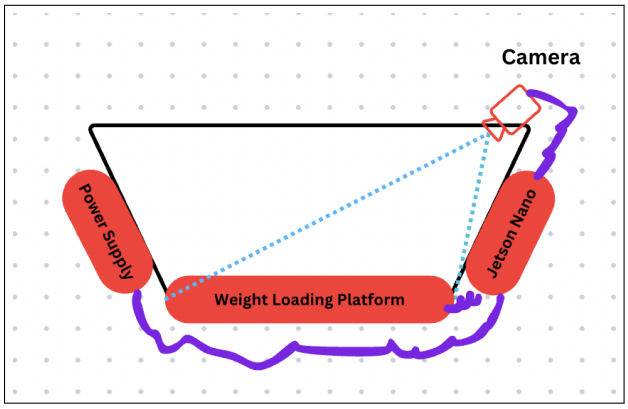

Fig. 3: RecipeCart Hardware Architecture Diagram

load cells, respectively. Along with the yellow shield pin, the E+, E-, A+, and A- pins receive input from the load cells [9]. On the other side of the HX711 PCB are the CLK and DATA gates, which are useful for regulating board operations and data transmission, respectively.

In building the weight scale system, we first solder the scale's data wires to the HX711's E+, E-, A+, and A- gates [4]. The VCC and GND gates are latched onto the Jetson Nano to provide power to the board. Finally, we connect the CLK and DATA gates to the Jetson Nano's pins 7 and 11, respectively, to enable data serialization.

Provided a successful linking process between the Jetson Nano and the HX711, we next configure the Jetson Nano with a custom driver to control the HX711 and interpolate the received data. We refer to a HX711 Python library on GitHub that builds on an existing HX711 codebase originally developed for the Raspberry Pi [2].

We then modify the source code to reflect updates in the *libgpiod* Python library used to control the Jetson's GPIO pins. We also adjust the reference unit according to the sensitivity of the kitchen scale—in our particular case, we found 320 to be the best estimated reference unit.

VII. Software Design and Implementation

The software design of the RecipeCart is characterized by two general processes: the machine learning architecture and the mobile app. This section details the procedure for setting up the AWS backend, examines the machine learning architecture, and provides the visual groundwork for the mobile app frontend design.

A. Amplify Backend

The AWS Amplify server maintains three natively configured services: an authentication system, a GraphQL-based API, and a set of Lambda functions.

The authentication system is managed by Amazon Cognito, which stores and provides the user credentials to the Flutter-based frontend. Credentials are hashed and access tokens are managed in Cognito, alleviating the user login and sign-up logistics. Any access to internal AWS resources is federated by Cognito, ensuring authorization and accountability with each API request and function call. Implementing Cognito also facilitates future integration with third-party social identity providers such as Google, Facebook, or Outlook. The authentication layer is added via `amplify add auth`.

A GraphQL-based API ensures data integrity when querying or mutating data stored in Amazon DynamoDB. The GraphQL API itself is managed and executed through AWS AppSync, which supports a subset of basic GraphQL commands and access patterns. The app's access patterns defines the GraphQL schema. Particularly, we are most interested in retrieving recipes given a set of input ingredients. This custom query must be directly handled via a Lambda resolver since AppSync does not natively support such a reversed filter search. The API layer is added via `amplify add api` menu.

```
65
66  type Settings @model @auth(rules: [
67    {allow: owner, operations: [read, update, delete]},
68    {allow: public, provider: apiKey}
69    ]) {
70    id: ID!
71    dietType: Int!
72
73    avoidances: [Ingredient] @hasMany(indexName: "bySettings", fields: ["id"])
74
75    cuisineTypes: [Int]
76    language: Int!
77    notifications: Boolean!
78    linkedDevices: [String]
79  }
```

Fig. 4: GraphQL Schema for the User Settings Data Type

AWS Lambda functions are integrated into the Amplify backend to handle the machine learning requests and responses. We employ three main Lambdas: one for submitting a photo of an ingredient to the object classification pipeline, one for controlling the barcode detection script on the Jetson Nano, and one for parsing the barcode and processing the product metadata. The functions are added and configured via the `amplify add function` prompt menu.

B. Storing and Populating Data in DynamoDB

The RecipeCart requires three DynamoDB tables: an ingredient table, a recipe table, and a user settings table.

The ingredient table is the crux of the RecipeCart app as it maintains connections with the Recipe and User Settings tables. The ingredient table stores both the user-owned ingredients and the ingredients required for each recipe. We distinguish the former by the presence of a *userID*—provided by Amazon Cognito Identity Pools—a barcode entry, and a *removed* status boolean, while the latter is denoted by the presence of a *recipeID* and *recipeName*. Additionally, the required quantity field for each ingredient reflects either the quantity currently owned by a user or the quantity needed in a parent recipe, respectively.

Each recipe in the recipe table has a name, a diet type, a cuisine type, a text block of instructions, and a list of ingredients. User-created recipes contain a *userID* entry. Each recipe also contains several fields to aid with implementing the recipe recommendation system such as the number of likes, the average rating, and the number of views.

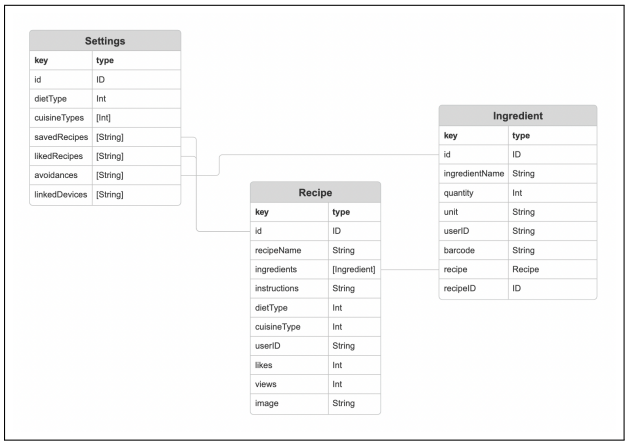

Fig. 5: Entity Relationship Diagram

Fig. 6: Lambda with Shell Script Commands

The user settings table contains all the user metadata not managed by Amazon Cognito—i.e. the non-credential data. The user's preferred diet type, cuisines, avoidances, and their saved and liked recipes. The *avoidances* field map to a list of *ingredientName* corresponding to ingredients that the user would like to avoid for personal or allergic reasons. The *savedRecipes* and *likedRecipes* fields contain a list of *recipeID*.

C. Weaviate Vector Database

A Kubernetes cluster is built in AWS via Kubernetes Operations (kOps). The kOps version must be compatible with the Kubernetes service on AWS as well as the Kubernetes command line (kubectl). The Kubernetes service in itself must be of version 1.23 to ensure compatibility with the Weaviate database and AWS. We then run the build script to install and configure the Weaviate vector database as a Kubernetes pod.

Once the Weaviate vector database can be reached via the Weaviate client, ingredient images are crawled via an open-source grocery dataset [3] and transformed into vector embeddings through Meta Dinov2. The vector database is then populated with the vector embeddings via the Weaviate client. These vector embeddings will be later referenced by the Weaviate client to classify incoming ingredient images from the mobile frontend.

D. Barcode Detection Pipeline

The barcode detection algorithm is loaded onto the Jetson Nano to simulate integration with a smart fridge, where the barcode detection is performed externally and subsequently forwarded to the RecipeCart backend server.

The entire process is initiated from the frontend via a Lambda function. The Lambda function sends a shell script command—containing the current user's Cognito Identity ID as a parameter—to the AWS Systems Manager Agent installed on the Jetson Nano to start or halt the barcode detection Python program.

Upon decoding the detected barcodes, the Python script packages the JSON result into a MQTT-based message to be published to a topic in AWS IoTCore—the topic name

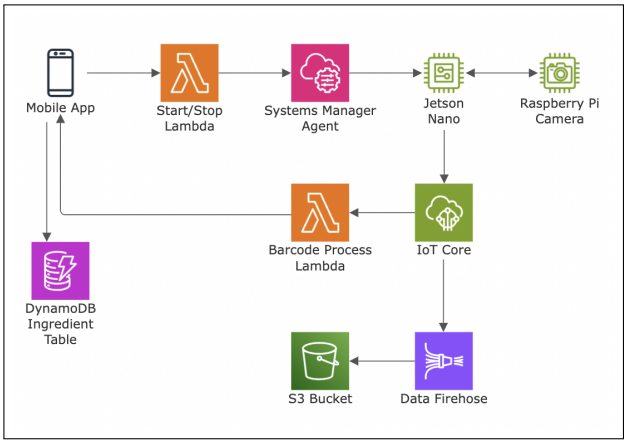

Fig. 7: Barcode Detection Pipeline Diagram

is determined by the Cognito ID of the user who initiated the barcode detection process. A Lambda function subscribes to the topic and consumes the MQTT messages. The Lambda function processes the barcode, checking if the ingredient product already exists in the DynamoDB ingredient table. If the product is not found, the Lambda must query and retrieve information from the external World Open Food Facts database. Otherwise, the ingredient is sent back to the frontend for display and user confirmation with the associated quantity and userID. The user may then choose to add the ingredient to their inventory or remove and rescan it.

In addition to the barcode-processing Lambda, we configured an Amazon Data Firehose sink that captures all the incoming MQTT messages in AWS IoTCore and dumps them into an S3 bucket for record-keeping and management.

E. Ingredient Object Classification Pipeline

This alternate approach to adding ingredients to the user inventory can be initiated via the mobile camera feature in the RecipeCart app. The picture is put into an S3 bucket which triggers a Lambda function to process the event.

The image pre-processing code, the Meta Dinov2 model, and the Weaviate client are packaged as a Docker container image, which itself is uploaded to Amazon Elastic Container Registry (ECR). From there, the Lambda function exposes a facet of the container image. Containerizing this pipeline not only allows for portability and resilience but is also necessary to allow the Lambda function to run the required PyTorch machine learning library; the PyTorch library is too large to be included into the Lambda function as a Lambda layer dependency.

Within the container, the input image is re-sized, cropped, standardized, and converted into tensor before being fed into the PyTorch-based Meta Dinov2 model. The Dinov2 model turns the image tensor into a one-dimensional tensor, which is then flattened to produce a vector embedding of the image.

The Weaviate client initiates a connection with the Weaviate cluster hosted on EC2 and queries it. The Weaviate host effectively performs an approximate K-Nearest-Neighbors (KNN) classification and returns a list of vector embeddings similar to the input image and their associated label. The labeled vector with the smallest KNN distance is returned to the mobile frontend as the classification result.

The user is then prompted to confirm the identified ingredient and enter the associated quantity. Upon confirmation, the ingredient is put into the DynamoDB ingredient table via the GraphQL API.

F. Recipe Recommendation System

The recipe recommendation system is divided into a filtering algorithm and a sorting algorithm. The filtering process is performed based on the user's diet type, cuisine type, and ingredient avoidances—information unique to each user stored in the settings table. The sorting algorithm employs the recipes' average ratings, its global number of likes, and its global number of views.

The ideal recommendation system takes into account the recency of the recipes' popularity metrics by tracking the time period in which certain recipes gain traction. Such additional fields would allow us to sort recipes based on metrics such as "most viewed within this month" or "most likes within the past week."

In light of the time constraints and project complications, we are implementing a simplified version of this content-based recommendation system that only looks at the global metrics.

VIII. System Testing and Results

Testing is performed along the implementation process of each component in the RecipeCart to ensure compatibility and integration with the rest of the system. All components of the RecipeCart have been tested individually, but integration testing to ensure end-to-end reliability is still underway.

A. Weight System

We first use a voltmeter to check for connectivity between the HX711 and the digital kitchen scale. Connectivity between the HX711 and the Jetson Nano is tested via the HX711 Python-based driver. The original HX711 driver code referenced an outdated *libgpiod* library, which initially cascaded into various dependency issues. This problem was mitigated after referring to other projects that forked from the main HX711 driver GitHub repository.

The driver's reference unit is then calibrated through trial and error. However, the weight values are relatively unstable and fluctuate between $\pm$ 20 grams, which inconsistently satisfies one of the engineering specification. The problem may be attributed to the HX711's inability to properly amplify the voltage values associated with weights at gram-level precision.

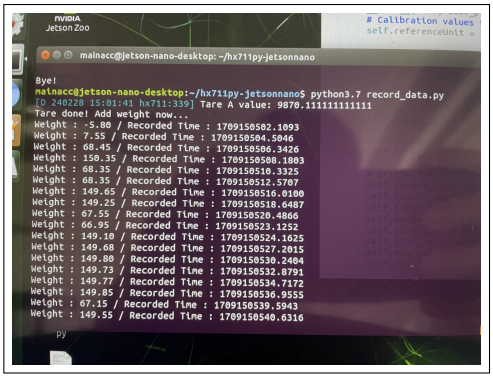

Fig. 8: Weight Values Retrieved in Jetson Nano

B. Barcode Detection

The barcode detection pipeline has been tested to properly fetch the decoded barcode from the Jetson Nano and processed in the Lambda function. The Lambda function can successfully query the World Open Food Facts database and retrieve the necessary metadata—if available. The Lambda can directly query and mutate the ingredient table, but we plan to have it invoke the GraphQL schema to ensure data integrity.

C. Meta Dinov2 and Weaviate Vector Database

The Weaviate Vector Database in the Kubernetes cluster was successfully tested for connectivity with the Weaviate client in the containerized Lambda. The image pre-processing and vector embedding through Meta Dinov2 was externally tested in a Jupyter notebook on Google Colab before being packaged into a Docker container and loaded as a Lambda function. The entire object classification pipeline by means of vector embedding reverse search works as a singular microservice and should be easily integrated into the rest of the RecipeCart system.

The object classification pipeline is satisfies two of the three main engineering specifications: it consistently achieves a classification accuracy of 85% and has a classification time of under 2 seconds.

D. GraphQL API

The GraphQL Schema has been tested for compatibility with DynamoDB but the associated API endpoints have yet to be fully configured and tested through the mobile frontend. GraphQL queries and mutations have been tested via AWS

App Sync but must also be tested from the frontend client perspective.

E. Flutter-based Frontend

The Flutter-based mobile frontend can be successfully loaded into any Android-based phone with a functional Cognito-based authentication system. The basic design skeleton is also developed to readily accommodate the GraphQL API and Lambda functions.

IX. Future Work

With the bulk of the machine learning completed and vetted, the main focus is currently to completely integrate the API to function with the mobile frontend, allowing for complex search operations.

The mobile frontend must fully reflect the different key features of the RecipeCart, including procedures for users to initiate the ingredient detection pipelines, view and manage their ingredient inventory, view and manage their saved recipes, explore the available recipes, and edit their food preferences.

The entire system has yet to be fully tested for end-to-end connectivity, and the three primary (highlighted) engineering specifications must be re-tested to ensure results reliability and consistency.

X. Biographies

Darren Ha is a Computer Engineering student at the University of Central Florida with interests in cloud computing and serverless systems. After graduating with his Bachelor's Degree in May 2024, he will continue exploring cloud technologies at Amazon Web Services as a Cloud Support Engineer.

Quan Nguyen is a senior student at the University of Central Florida with interests in full-stack development and backend server deployment and administration. With his Bachelor's Degree in Computer Engineering, he plans to join Epic Systems as a Client System Administrator.

Bryan Ha is a senior Computer Science student at the University of Central Florida with a passion for machine learning and computer vision. His research-oriented goals drive him to further pursue an advanced degree specializing in AI/ML.

Darren Ha

Quan Nguyen

Bryan Ha

Acknowledgment

We would like to thank Dr. Enxia Zhang for her involvement in aiding the team in the later stages of the project's development.

We would also like to acknowledge Dr. Lei Wei for his assistance and understanding with the RecipeCart team. His constructive inputs provided the team with an insightful approach with handling certain engineering specifications.

References

[1] AWS Amplify, "AWS Amplify." Retrieved from aws.amazon.com/amplify/

[2] I. Kempei, "HX711 for Jetson Nano," March 2022. Retrieved from github.com/kempei/hx711py-jetsonnano

[3] M. Klasson, "Grocery store dataset," November 2019. Retrieved from github.com/marcusklasson/GroceryStoreDataset

[4] I. Luuk, "50kg load cells with HX711 and Arduino. 4x, 2x, 1x diagrams," April 2020. Retrieved from circuitjournal.com/50kg-load-cells-with-HX711

[5] Meta, "DINOv2: State-of-the-art computer vision models with self-supervised learning," April 2023. Retrieved from ai.meta.com/blog/dino-v2-computer-vision-self-supervised-learning/

[6] Nvidia, "Jetson Nano." Retrieved from developer.nvidia.com/embedded/jetson-nano

[7] Particle Community Troubleshooting Blog, "Strange behavior of load cell HX711," May 2019. Retrieved from community.particle.io/t/strange-behavior-of-load-cell-hx711/49566/2

[8] Raspberry Pi, "About the camera modules." Retrieved from www.raspberrypi.com/documentation/accessories/camera.html

[9] N. Seidle and A. Wende, "HX711 Load Cell Schematic," SparkFun, May 2019. Retrieved from cdn.sparkfun.com/assets/f/5/5/b/c/SparkFun_HX711_Load_Cell.pdf

[10] Weaviate, "Welcome to Weaviate docs." Retrieved from weaviate.io/developers/weaviate

[11] World Open Food Facts. Retrieved from world.openfoodfacts.org/