AI-Based Mobile Recommendation System for Food Purchases (Report)

Submitted as part of the requirements for: CE903 Group Project

Supervisor: Dr. Somdip Dey

Group 17

Lee Socretquuliqaa, 2204551

Omkar R Kharkar, 2205233

Lamis Kheiralla, 2200345

Waleed Bin Khawar, 2205265

Syed Daud Kazmi, 2205008

Meena Kohli, 2201597

Veeramani Mahadevan, 2201166

Ancy Mathew, 2200602

Ronak Punjabhai Karmur, 2202285

Paulius Uliackas, 2201498

Versions: 4 **Date: (22/03/2023)**

Abstract

AI models are now used in most industries such as food, weather, traffic, etc. to predict behavioural outcomes. As part of this team project, we have collaborated with Nosh Technologies, a deep tech company in the UK specialising in food management, to predict food purchase by customers. The purpose of the project is to deliver a working recommendation system in a mobile application for food purchase using AI methodologies such that food products can be offered to new/existing customers of the company based on the purchase history of existing customers on the mobile application.

Contents

List of Figures

List of Tables

1 Introduction

AI has become a widely adopted technology that powers many real-world applications ranging from food, weather, traffic, etc. An important application of AI is recommendation systems. In the food industry, platforms that offer groceries items often have hundreds of products that the user can choose from. Recommendation systems play an important role in helping users find the right products from a vast catalogue of options. It provides a level of personalising that can greatly improve the user experience on a platform.

Development of recommendation systems centres around modelling the user's preferences using either their previous interactions with the system or their characteristics in relation to the items in the systems. It often involves learning a function that can predict the utility rate of a user on a particular item **Ricci.etal2015a**. In the context of food purchase recommendations, the task would involve learning to predict how likely a customer would buy a particular product. Approaches for food purchase recommendations are taken from the broader research into recommendation systems. It can be grouped into three camps: (i) Content-Based, (ii) Collaborative Filtering, and (iii) Hybrid. Regardless of the approach, they are often developed using standard benchmark datasets **Ricci.etal2015a**. In the real world, these data are often costly to collect for privacy reasons or a lack of a user base.

To tackle this issue, we propose a method for generating recommendation data for food purchase recommendations based on Word2Vec and NGram. We trained and tested the generated data with a novel recommendation approach called Feature Faking and a GNN based on GraphSage **Hamilton.etal2017**. We demonstrated that our synthetic data generation process matches the distribution of real buying patterns, and our Feature Faking method outperforms our baselines and GNN-based approach in Accuracy@10, Precision@10, and AP@10. We collaborated with Nosh Technologies to deploy our recommendation in a mobile application for food purchases. We built the application using Flutter for cross-platform support on iOS and Android and deployed our recommendation engine on a Flask backend to food products to new/existing customers of the company based on the purchase history of existing customers.

This document is structured as follows. First, Section 2.1 to Section 5.5 discusses the background, related work, methodology, system design, implementation, experimental setup, Testing and result and discussion of our experiments for developing a novel synthetic data generation process and recommendation approaches. Based on our findings, we detail our reasoning behind choosing GNN-based recommendation for usage with Nosh Technology in Section 5.6 .

2 Background and Related Work

The main focus of this project is to develop a working recommendation system for food purchase that will be deployed in a mobile application. It involves understanding the data needed and the approaches that are available. This section provides background for understanding these concepts as well as the related work concerning this project. First, it discusses the data needed to train a recommendation engine. Then, it discusses what synthetic data are.then, it discusses an overview of the different recommendation approaches.Finally, it discusses literature review related to this project.

2.1 Related Work

The associated research is discussed in the paragraphs that follow, and the background of the food recommendation system is further explored.

The literature surrounding food purchase recommendations falls into the three categories of (i) Content-base approaches, (ii) Collaborative Filtering, and (iii) Hybrid Approaches. Ueda et al. **ueda2011user** and Zitouni **zitouni2022new** introduces a collaborative filtering-based recommendation system for food recipes by utilising ingredients from food items to construct users' ingredient preferences. In the Content base group, Bianchini et al. **bianchini2017prefer** and Sedlak **sedlakcontent** utilise food information such as cuisine, cooking method, ingredients, and nutritional values to calculate similarities between items. However, these approaches suffer from traditional over-specialisation and cold start problems leading to low accuracy scores. The Hybrid approaches utilise Content base and Collaborative filtering in different scenarios through the system. Both Kumar **kumar2016survey** and Li **li2018application** reported an increase in accuracy for their Hybrid Approaches. These related works show that a hybrid approach can improve the accuracy and relevance of food product recommendations by combining multiple recommendation techniques. However, the effectiveness of the hybrid approach depends on the quality and completeness of the data used to represent the food products and user preferences, as well as the choice of algorithms used for collaborative and content-based filtering.

The work outline in this section shows that hybrid-based recommendation works well for food purchase recommendation. It also shows that there is a gap in using synthetic data for building food recommendation systems. To fill in this gap, we propose a method for generating recommendation data for food purchase recommendations based on Word2Vec and NGram. We trained and tested the generated data with a novel recommendation approach called Feature Faking, and a GNN-based approach on GraphSage **Hamilton.etal2017**.

2.2 Recommendation System Data Sources

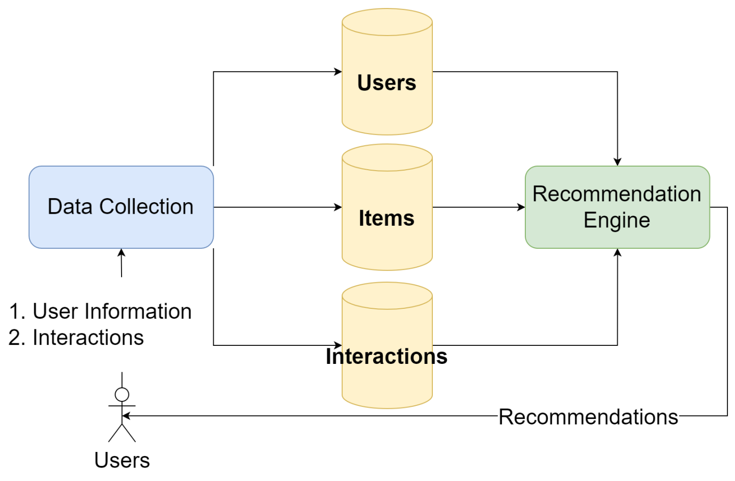

Figure 1: Recommendation Data Sources

There are three data sources that a recommendation engine draws from (i) Users, (ii) Items, and (iii) Interactions **Ricci.etal2015a**. The users refer to the entity that a recommendation engine has to provide suggestions to while the items are entities being recommended to the users. These data are often expressed by their characteristics. For instance, a system may collect information about the user's name, age, etc. Similarly, when adding a new item to the system, it may record information about its price, category, etc. The interactions are the links between the users and items. This can be in the form of purchases, views, likes, etc. It forms a historical representation of a user's preferences. These three data sources are often collected explicitly by asking the users or implicitly by inferring them from the user's activities in the system. However, due to a lack of a user base or privacy concerns, they are often hard to collect. Therefore, synthetically generating these three data sources emerges as an interesting alternative.

2.3 Synthetic Data

Synthetic data generation involves creating fake data that are statistically similar to real data. It is popular in the field of Computer Vision but has practical implications in other domains as well. Synthetic data generation yields two major benefits. First, it is cost-effective. It allows for faster and cheaper data accumulation. A program can generate thousands of samples in minutes for use in the training pipeline. Second, it aids with privacy. Some type of data is sensitive and cannot be collected. Therefore, being able to generate samples that are similar to the real data provides opportunities to build learning models for that tasks that otherwise would be prevented for privacy concerns. Methods for generating synthetic data rely on building a base distribution/samples that the generation algorithms/models can draw from. This process still holds true for the current Generative adversarial network (GAN) based approaches. In the context of food recommendation, a base distribution of purchase data will need to be created before synthetic data can be generated and trained on the different recommendation approaches.

2.4 Recommendation Approaches

The approaches for generating recommendations can be grouped into three main categories: (i) Content Based, (ii) Collaborative Filtering, and (iii) Hybrid **Ricci.etal2015a**. Content-Based engines rely on the features of the users and the items to produce recommendations. It often involves building a classifier that can predict the ratings that a user will give to a particular item based on the features of the item, the current user, and the ratings the user has given similar items in the past. Collaborative filtering engines, on the other hand, rely on information from other users to provide recommendations for the current user. A user-item matrix where each cell corresponds to the ratings a user has given to similar items in the past. The global rating information is then used to recommend items to the current user based on other users who have similar rating patterns in the system. Both Content-Based and Collaborative Filtering engines have drawbacks. Content-Based engines often lead to the issue of specialisation where the users will only be shown items that they have seen in the past. Collaborative Filtering engines suffer from the issue of Cold-Start when a new user has been added to the system or does not have enough rating information. To overcome this problem, a Hybrid approach that combines Content-based and Collaborative Filtering into one system is often introduced. In this approach, Content-based recommendation is shown to users when their rating information is scarce before transitioning to Collaborative Filtering recommendations when they provide more interactions. Overall these three approaches are used by popular algorithms in recommendation system research and are widely adopted in many real-world systems **Eksombatchai.etal2017**.

3 Methodology, System Design, and Implementation

3.1 Methodology

This section discusses the methodology followed for synthetic data generation for purchase data, products data, and users data. As well as the methodology of the different recommendation approaches (graph-based, feature faker, and auto-encoder).

3.1.1 Synthetic Data Generation

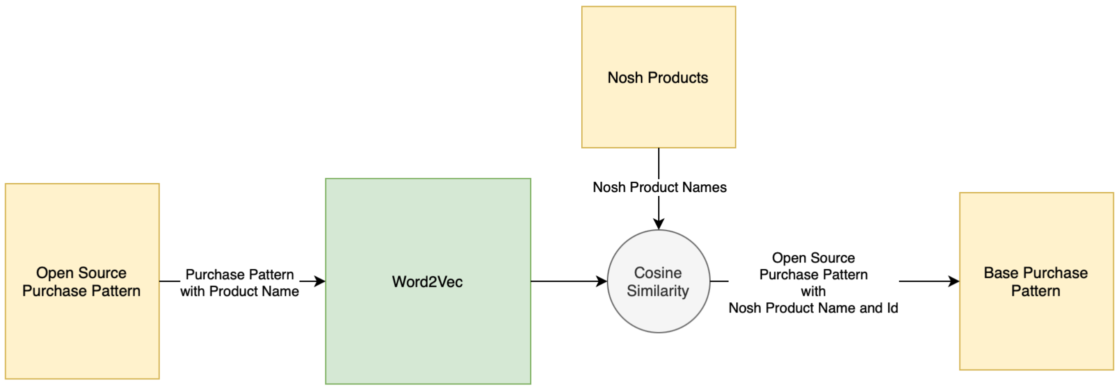
Nosh Products
Nosh Product Names
Open Source Purchase Pattern
Purchase Pattern with Product Name
Word2Vec
Cosine Similarity
Open Source Purchase Pattern with Nosh Product Name and Id
Base Purchase Pattern

Figure 2: Base Purchase Data Generation

Generating synthetic data for food purchase recommendations involves generating three main data aspects of a recommendation engine: (i) Users, (ii) Products, and (iii) Purchase Data. The methodology adopted for this project first generates the purchases and product data before generating the user data. The remaining of this section outlines the details of each step.

3.1.2 Purchase Data Generation

The main goal of generating the purchase data is to have the buying pattern of the generated data matches the actual buying pattern of the real data. By ensuring that the buying pattern follows the same trend, the generated data should possess the underlying characteristic of the actual data. This is essential because learning models trained on the generated data need to generalise to the actual buying patterns of the customer. To generate purchase data, this project took inspiration from Natural Language Processing. First, it relies on a pre-trained Word2Vec word embeddings **mikolov˙efficient˙2013** to match the names of products in an open-source food purchase data **noauthor˙groceries˙nodate** with the nosh products using cosine similarity. The goal is to build a base product purchase distribution. The purchase data provided by Nosh Technologies contains the buying patterns of only two customers. Hence, the additional buying patterns can be taken from open-source data. With the addition of open-source data, the base purchase pattern contains 38765 purchase sequences. Then, using the base purchase distribution, the following process is performed to simulate a purchase sequence: (i) Randomly pick a starting product and the number of items bought using the base purchase distribution. (ii) Use a 4-Gram model to predict the next item purchase based on the previous three purchases. (iii) Repeat steps (i) and (ii) until the number of items is reached. The process is then repeated to create purchase data for 400 customers, each with 1000 purchase sequences. Figure 2 illustrates this process in the form of a flow chart.

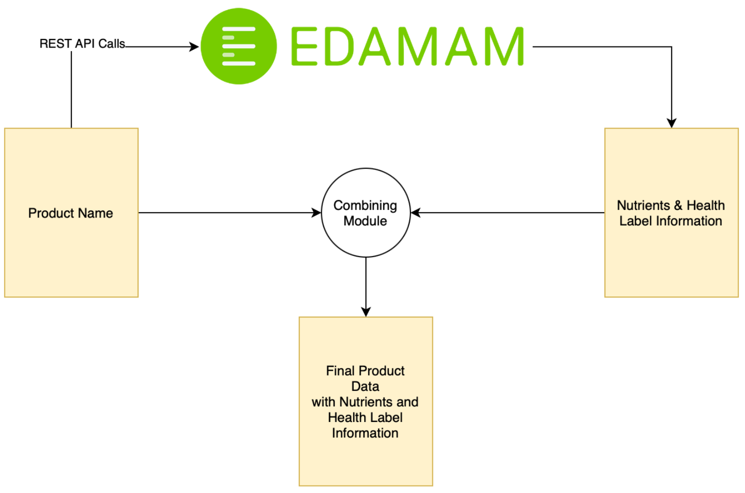
REST API Calls
EDAMAM
Product Name
Combining Module
Nutrients & Health Label Information
Final Product Data with Nutrients and Health Label Information

Figure 3: Products Data Retrieval

3.1.3 Products Data Generation

The product data provided by Nosh Technologies contains 207 unique products. Each product consists of four pieces of information: (i) id, (ii) name, (iii) shelf life, and (iv) storage. This information alone does not provide a descriptive characteristic of a product. Our view is that people buy a product based on: (i) The food they are making. (ii) Their current dietary preferences. Hence, each product should be associated with these features. To accomplish this, a retrieval process is performed using Edamam **noauthor˙edamam˙nodate**, a food database with free-to-use API. With this API, nutrients (Energy, Protein, Fat, Carbohydrate, and Fiber), calories, and health labels (Fat Free, Kato Friendly, etc) are retrieved for each product. Furthermore, each product is also linked to different recipes. To do this, an open-source recipe data set is used **noauthor˙recipe˙nodate**. Each recipe contains the type of cuisine it is and a list of ingredients associated with that recipe. The same Word2Vec approach is used to match the names in the ingredient lists with nosh products.

3.1.4 Users Data Generation

To generate user data, information from both the generated purchase data and product data is used. It links back to our view that user buys food products based on their dietary preferences and as part of the recipe that they are planning to make. Hence, for each of the 400 customers created during the generation of the purchase, their dietary and health label information is inferred from their purchase data. To achieve this, we calculate the average of how many times a user purchases a product with specific health labels and cuisines.

3.1.5 Recommendation Approaches

To validate the usability of our synthetically generated data we trained different recommendation approaches on it. This section discusses those approaches which are (i) the Graph-Based Recommendation Approach, (ii) Our Novel Approach (Feature Faker), and (iii) the Auto Encoder approach.

3.1.6 Graph Based Recommendation

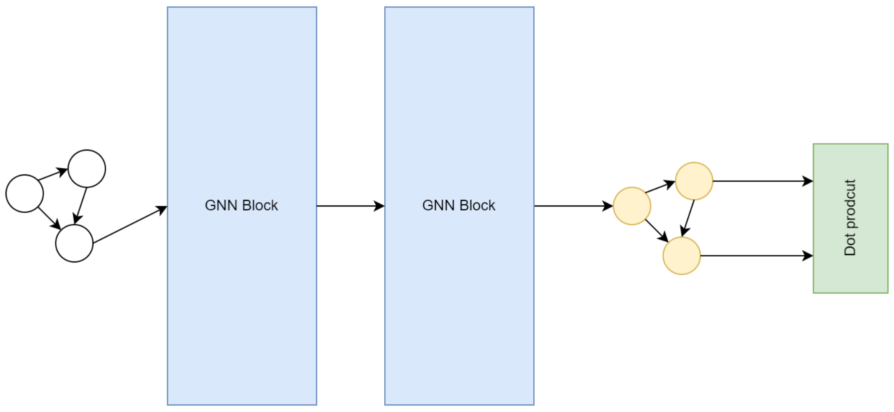

Figure 4: Graph Network Neural Architecture. Each GNN block is a GraphSage-based block.

One recommendation approach adopted for this project is a graph-based approach. The reasoning behind this decision is that Graph Neural Networks (GNN) have been widely adopted for the task of recommendation due to their ability to leverage the spatial and temporal information of the graph data. To tackle the task of recommendation with GNN, it can be framed into a Link Prediction task where the goal is to predict if a link exists between a user node and a product node.

The variant of GNN that this project is based on is GraphSage **Hamilton.etal2017**. The key element in GraphSage is that it can learn embeddings of a node based on just the neighbourhood information. It frees the learning process from relying on knowledge of the entire graph. This resulted in an inductive learning approach that does not have to be retrained when adding a new node. Furthermore, these latent features capture the collaborative nature of recommendation data as well as the characteristics of the users and the items. Therefore, the recommendation engine can utilise both of these embeddings to provide either Content Base or Collaborative filtering type of recommendations.

Figure 4 outlines the architecture of the GNN. We employ two GraphSage layers to learn the two hop hidden features of each node. Then, we use binary cross-entropy to minimise the dot product between the user and the product embeddings.

$$\mathcal{L}_{BCE} = -\frac{1}{|\mathcal{D}|} \sum_{(u,v)\in\mathcal{D}} y_{u,v} \log \hat{y}_{u,v} + (1 - y_{u,v}) \log (1 - \hat{y}_{u,v}) \tag{1}$$

where:

- $\mathcal{D}$: the set of edges in the graph.
- (u, v): an edge between nodes u and v.
- $y_{u,v}$: the edge label
- $\hat{y}_{u,v}$: the dot product between the user and the product embedding.

3.1.7 Feature Faker

Feature faker is a recommendation algorithm that links products and users with weighted connections. The algorithm is called feature faker because it is unknown what the actual connections represent. When the model is initialised each product gets assigned x number of random weights, which represent what features the product contains. The customer gets assigned x number of even weights, they represent what features the user likes. When a customer purchases a product, the customer's features are adjusted based on the product's features. The product's features remain unchanged. The algorithm then recommends items which have the most similar features to the users. This is measured using the Manhattans distance.

If we consider examples such as:

Items	Feature 1	Feature 2	Feature 3	Feature 4
product 1	25	25	25	25
product 2	10	10	10	70

The table shows that product 1 is linked to each feature by 25 per cent. But product 2 is heavily influenced by feature 4. Costumers(users) are represented in a similar way as products are. Except, the customers are not linked to each other's features but are linked based on the products they bought. Each customer's feature cell in the table represents how much that customer likes the feature.

Users	Feature 1	Feature 2	Feature 3	Feature 4
customer 1	20	20	40	20
customer 2	0	50	0	50

The customer's features are adjusted based on the items that they have bought. The adjustment formula is customer feature + product feature. For example, if customer 1 buys item 2:

Users	Feature 1	Feature 2	Feature 3	Feature 4
customer 1	20 + 10 = 30	20 + 10 = 30	40 + 10 = 50	20 + 70 = 90
customer 2	0	50	0	50

Finally, the features are represented based on the percentage thus features for customer 1 have to be scaled down based on the formula: customer feature * 100 / sum of all customer's features. We end up with:

Users	Feature 1	Feature 2	Feature 3	Feature 4
customer 1	30 * 100 / 200 = 15	30 * 100 / 200 = 15	50 * 100 / 200 = 25	90 * 100 / 200 = 45
customer 2	0	50	0	50

This process runs for each interaction of the user. The approach has advantages, such as working with limited data. On the other hand, the algorithm can be improved by adding a search method for optimal product feature correlation between products, the reason is that the algorithm might be sometimes unstable, due to the product random initialisation. Secondly, the algorithm should receive some noise, to produce a few random suggestions for the user once in a while. Which is to be implemented.

3.1.8 Auto-Encoder

In the context of content-based recommendation systems, Auto-encoding variational Bayes (AEVB) **kingma2013auto** can be used to generate new items that are similar to the existing ones. As AEVB, the encoder maps the input data to a probability distribution over a low-dimensional latent space, and the decoder maps the latent variables back to the input space. This allows for the reconstruction of the input data and the generation of new samples by sampling from the learned distribution over the latent variables.

AEVB can also be used to improve the interpretability of content-based recommendation systems. By visualising the learned latent space, it is possible to identify clusters of similar items and understand the key features that distinguish them. This can help content creators and marketers to understand their audience better and create more targeted and relevant content. AEVB is a powerful tool for content-based recommendation systems that can enhance the quality and diversity of recommendations and provide valuable insights into the underlying structure of the data. Overall, the approach followed by AEVB combines the strengths of autoencoders with the flexibility of probabilistic modelling to learn a powerful generative model that can be used for a variety of tasks, including content-based recommendation systems and data generation.

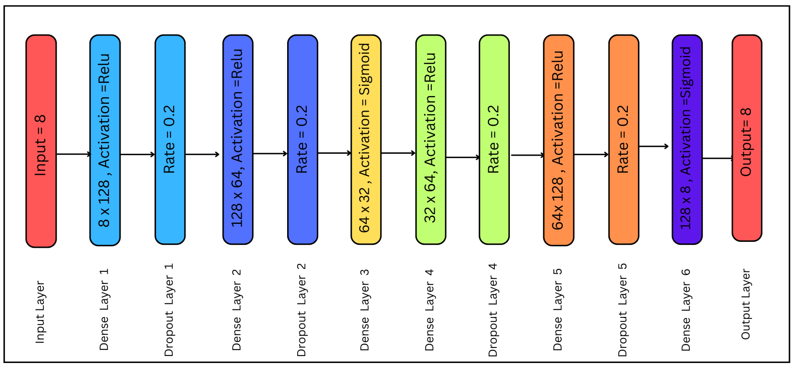

Figure 5: Model Summary

Figure 5 represents the model summary of the auto-encoder. The input size for this model is 8, and it is then passed on to the first dense layer (128 x 8), where the activation function is Relu. The dropout layer follows, and it randomly removes some of the units during training to avoid over-fitting **pmlr-v28-wan13**. Once the output from the preceding layer has been received, it is again sent to a second dense layer (128 x 64) that serves as relu's activation layer before moving on to the dropout layer. The process till this point is known as the Encoder part. The Fourth Dense layer is called the Latent part with activation function sigmoid **Han˙1995**. At this point, the data is converted to low-dimensional latent space. After this decoder part begins, the process repeats itself until dense layer 6 as indicated in the diagram, however, at this step, the activation function switches from relu to sigmoid. This is being done in order to restore the data to its original state.

$$\text{Sigmoid Function} = \mathrm{S(x)} = 1/1 + \mathrm{e}^{-\mathrm{x}} \tag{2}$$

where,

- e = Euler's number

$$\text{Relu Function} = f(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

where,

- x = the input to the neuron

$$\text{Dense Layer} = \text{activation(dot(input, kernel) + bias)} \tag{4}$$

where,

- Input represents the input data
- Kernel represents the weight data
- Dot represent numpy dot product of all input and its corresponding weights
- Bias represents a biased value used in machine learning to optimize the model
- Activation represents the activation function.

3.2 System Design

This section outlines the requirements that were listed in the Requirements Specification Documents **srsProject17**. We repeat the information for quick reference and we elaborate in more detail about the use cases, system architecture, data flow diagram, and er diagram.

3.2.1 Requirements

Based on discussions with Nosh Technology, we have gathered the below requirements.

3.2.2 Recommendation Engine

- The engine shall be trained on the synthetic data of 200+ customers.
- The engine shall recommend products with similar characteristics.
- The engine shall recommend products based on other users that share similar interests.
- The engine may recommend frequently bought together items.

3.2.3 Mobile App

- The app shall have a sign-up page.
- The app shall have a login page.
- The app shall allow the user to search for products.
- The app shall have a page to display all products.
- The app shall allow the user to add products to the cart
- The app shall allow the user to check out the items they have in the cart
- The app shall allow the user to view the product details
- The app shall display recommendation to new users
- The app shall display to the users the reason of recommendation.

3.2.4 Backend System

- The system shall allow users to create an account.
- The system shall be able to record purchases.
- The system shall provide recommendation for all users.
- The system shall record the products added to the user's cart.
- The system shall allow users to search for products.
- The system shall recommend products that are in stock.
- The system shall allow admin to add new items to the store.

3.2.5 Performance Requirements

Static numerical requirements

- The mobile app shall support both the iOS and Android platforms.
- The backend system should run on a demo server with a capacity of: 1GB Memory, 1 vCPU, 1,000 GiB Transfer, 25 GiB SSD provided by Digital Ocean .

Dynamic numerical requirements

- The recommendation engines shall achieve an Accuracy@10 metric higher than random.
- The backend system shall return recommendation results in under 1 second.
- The mobile app should not take more than 10 seconds to load the initial screen.
- The application should support 1000 simultaneous user.
- The application should handle user orders of food items 100 requests per second.

3.2.6 Use Cases

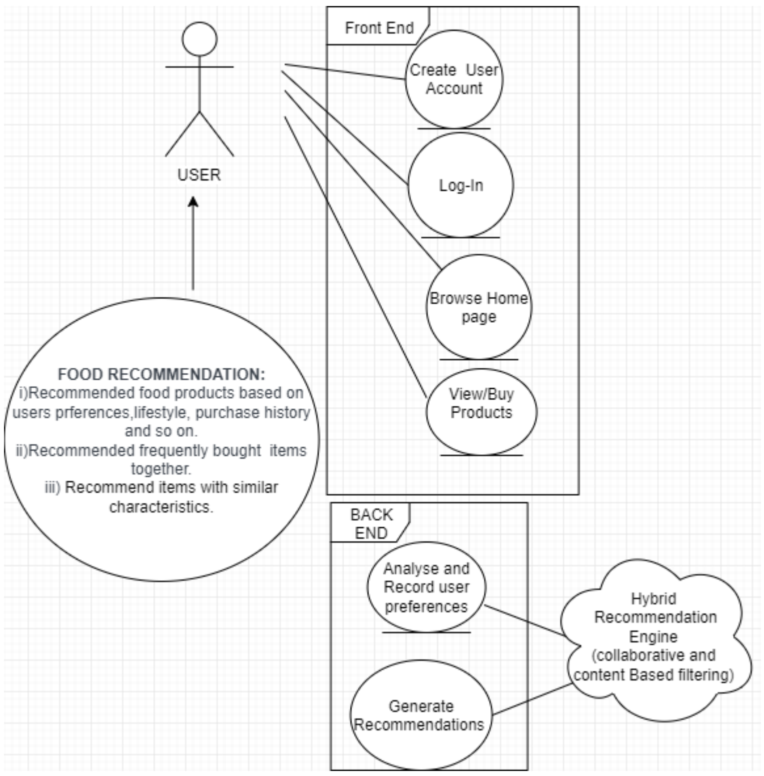
USER
Front End
Create User Account
Log-In
Browse Home page
View/Buy Products
FOOD RECOMMENDATION:
i)Recommended food products based on users prferences,lifestyle, purchase history and so on.
ii)Recommended frequently bought items together.
iii) Recommend items with similar characteristics.
BACK END
Analyse and Record user preferences
Generate Recommendations
Hybrid Recommendation Engine (collaborative and content Based filtering)

Figure 6: Use Case Diagram

From the requirements gather in the previous section, we developed the following use cases.

3.2.7 Front-End

The user interface or front end is the part of the system that users interact with to view food products and receive personalized recommendations. The user interface typically consists of several components, including a signup page, a home page that displays food products, and a search bar that enables users to search for specific products.

Signup Page:

The signup page is the first component of the user interface that users interact with when they join the system. The signup page typically requires users to provide basic information such as their name, email address, and password. Once users have successfully created an account, they can access the system's features and personalized recommendations.

Home Page:

The home page is the main component of the user interface that displays food products and personalized recommendations. The home page typically includes a variety of food products, including popular and trending items. The recommendations on the home page are personalized to the user's preferences and purchase history.

Search Bar:

The search bar is a component of the user interface that allows users to search for specific food products. The search bar should be easy to use and provide accurate results based on the user's search query.

Cart:

The cart is a component of the user interface that displays the food products that the user has added to their cart. The cart should be easy to access and allow users to modify their order or checkout easily.

3.2.8 Backend

The back-end system is the part of the system that performs the data processing and analysis necessary to generate personalized recommendations for the users. The back-end system typically consists of several components, including data storage, recommendation algorithms, and machine learning models. The back-end system stores users' purchase histories, cart histories, and bought-together items. This data is used by the recommendation engine to generate personalized recommendations. The data storage system should be designed to ensure data privacy and security.

3.2.9 Recommendation Engine

The recommendation engine is the core of the system; it analyzes users' preferences and purchase history to provide personalized recommendations. The recommendation engine typically consists of several components, including collaborative filtering, content-based filtering, and hybrid recommendation methods.

Collaborative Filtering

It is a recommendation algorithm that analyzes users' purchase histories and identifies patterns in their behavior. These patterns are used to recommend products that similar users have purchased. Collaborative filtering methods are typically based on user-item interactions, where users are matched with similar users based on their shared preferences.

Content-Based Filtering

It is a recommendation algorithm that analyzes the attributes of food products, such as nutritional information, ingredients, and cooking method, to recommend products that are similar to those that the user has purchased. Content-based filtering methods are typically based on item-item interactions, where products are matched with similar ones based on their shared attributes.

Hybrid Recommendation Methods

This methods combine collaborative filtering and content-based filtering methods to provide more accurate and relevant recommendations. These methods typically use machine learning algorithms, such as neural networks or decision trees, to analyze users' purchase histories and generate personalized recommendations.

3.2.10 System Architecture

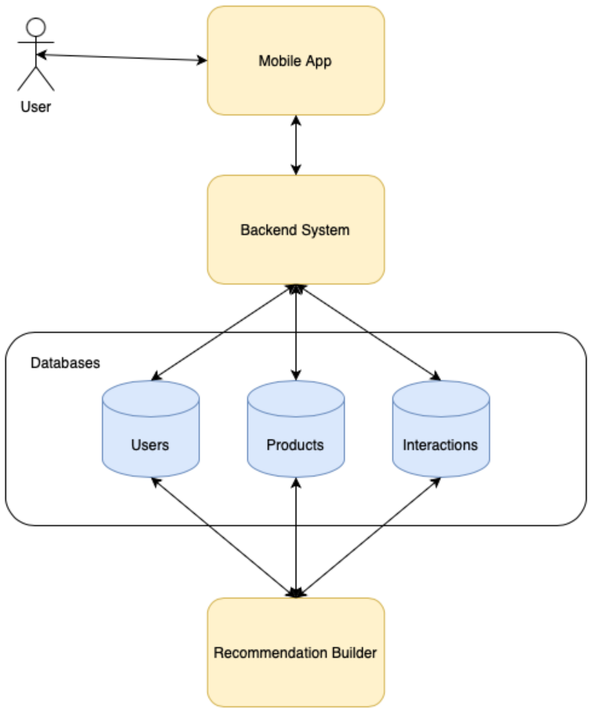

Figure 7: System Architecture Diagram

The system architecture of a food recommendation system mainly consists of four components. Mobile application is used to interacts with the user, where user can create their account, browse different products, add items to the cart and purchase the products, and based on their preferences receive recommendations. The logic of the system is built at the backend and is developed to communicates with the frontend for receiving and sending information to the mobile application and updates database accordingly. The database store relevant data regarding users, products, and interactions. The relational database is used, and it creates data files which is further used to recommend products on the front end. Recommendation builder component is actually the heart of the system architecture. It use hybrid recommendation algorithm to provide more accurate recommendation to the application users. These interfaces can be implemented using various technologies and frameworks. In this project no hardware and system interfaces are required.

3.2.11 Data Flow Diagram

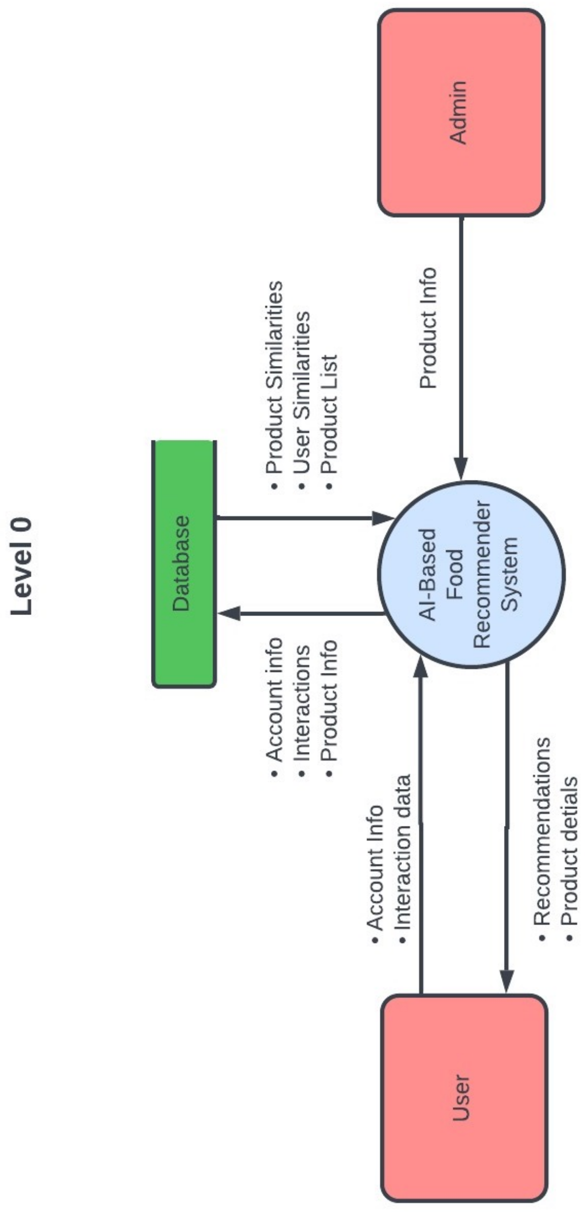

Figure 8: Level 0 Data Flow Diagram

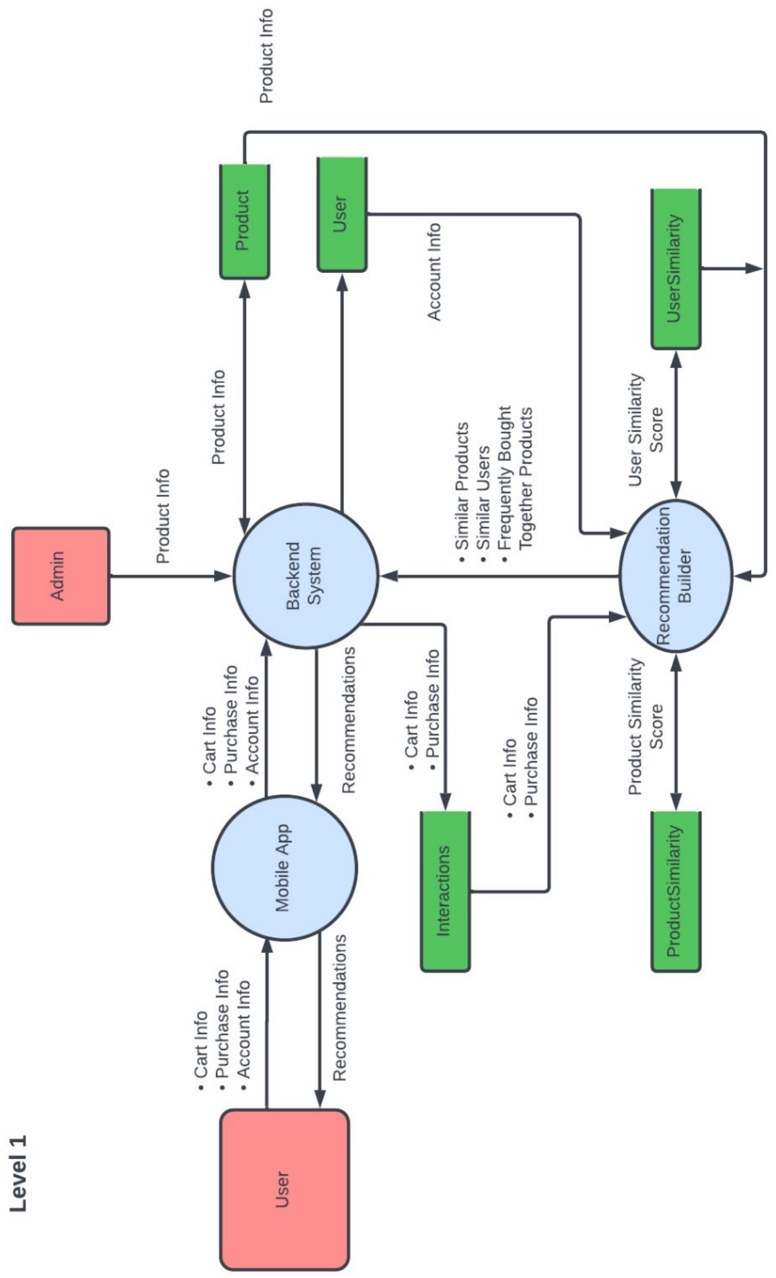

Figure 9: Level 1 Data Flow Diagram

The main entry point of the data into our system is through the mobile application. The user's input information such as login details and product interactions are recorded to the backend system. The backend system then gets the recommended products from the recommendation based on user details and product interactions. The backend system acts as a central repository that interacts with the mobile app and recommendation system. It is essentially the database that receives cart information, purchase information, and account information from the mobile app and sends the recommendation back to the mobile app. The backend system also sends the user interaction details and cart information to the recommendation system and receives similar products and frequently bought products for the user. Finally, when the data is stored in our database, the recommendation engine utilises those information to be the brain of the entire recommendation system. It receives user details, product information, and user interaction details from the backend system and sends the recommended products based on frequently bought together and product similarity for the user.

3.2.12 Entity Relationship Diagram

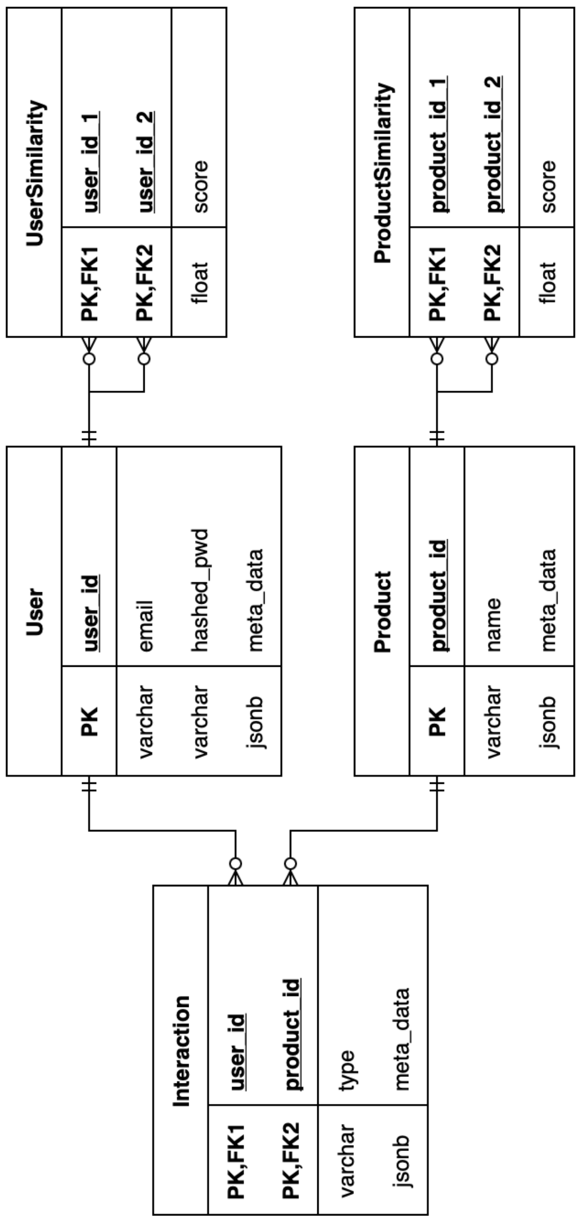

Figure 10: Entity Relationship Diagram

Figure 10 depicts the entity relationship for different components of the food recommendation system. The user will be the active agent in this system so the user needs a unique customer ID, a name, an email address and a password. The product data has attributes product IDs and metadata like quantity, shelf life, nutritional value etc. The diagram introduced the relationship of the user with the product. The customer adds products to the shopping cart. The relationship contains the cardinality idea due to the fact that a consumer can add one product at a time to their cart and that each cart can be linked to a single client.

The two new entities derived from user and product are user similarity and product similarity. The recommendation algorithm creates a user similarity table based on food preferences for a user with other users. The two different users with the most similar choices are at the top of the table. The product similarity entity is derived from the product table. Two products with similar characteristics will have a higher value in the product similarity table. Further, the cosine similarity is used in the interaction table to recommend user preferences.

3.3 Implementation

This section details about the implementation of a complete recommendation system for Nosh Technology based on the requirements outline from Section 3.2. It first discusses the technology and tools used for building the Mobile Application and the Backend System. Then, discusses the implementation the mobile application and backend system individually. It ends with an overview of the code base.

3.3.1 Technology and Tools

Technology/Tools	**Version**
Python	3.11.1
Flask	2.2.3
Flutter	3.7.1

Table 1: Development Tools and Versions

The technology used to build the project can be grouped into two areas. For the mobile application, we rely on Flutter. The reasoning behind this decision is that Flutter offers us cross-platform compatibility. The developed application can be run on both iOS and Android. This gives Nosh Technology fewer code bases to maintain while broadening access to the application for users on different platforms. For the backend, we rely on Flask with a Postgres database. We chose this combination for backend development for two reasons. First, we developed, trained, and tested our recommendation approaches using Python. Therefore, Flask is our framework of choice because it also runs on Python. It means that we can easily take our inferences code for deployment on the backend. Second, although a graph-based database like (Neo4J **noauthor˙neo4j˙nodate**) would be more efficient for deploying GNN models, the team's familiarity with SQL-type databases allows us to be more efficient during development.

3.3.2 Client Server Model

The communication between the Flutter Mobile Application and the Flask backend is done through REST API calls. We employ SHA256 encryption for encrypting user passwords before storing them in our database.

3.3.3 Mobile Application

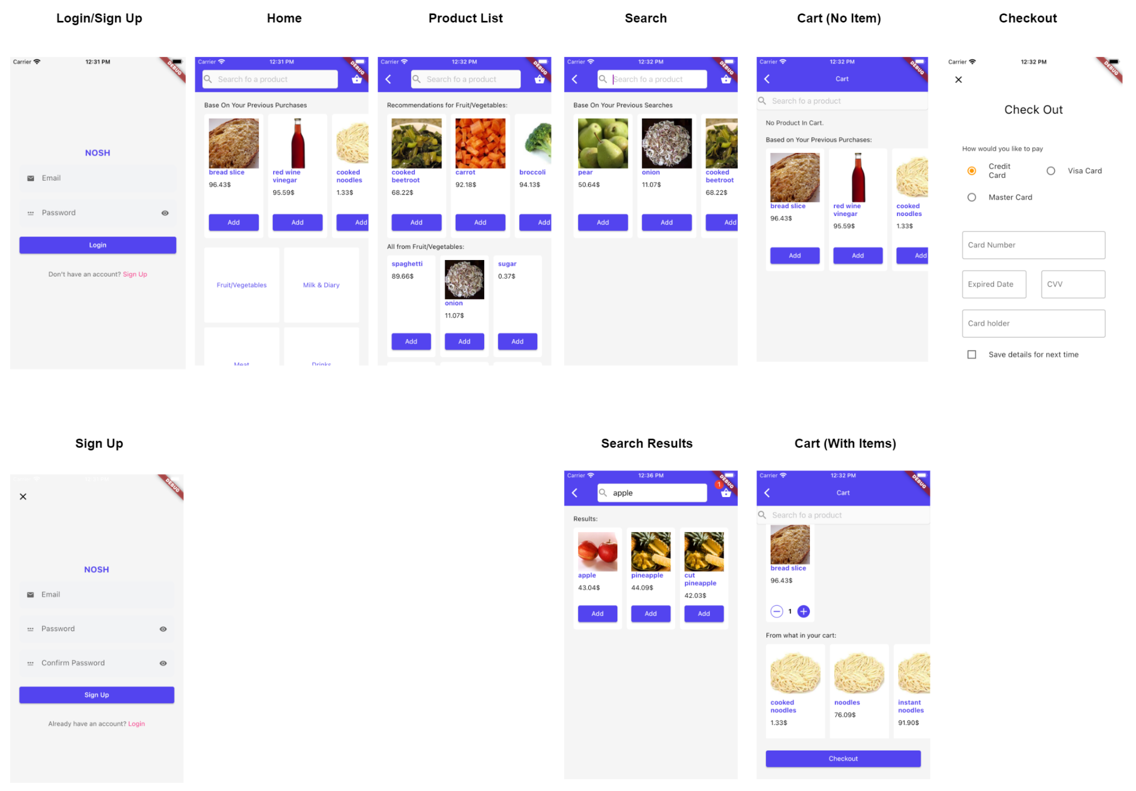

Figure 11: User Interfaces of Mobile Application

We developed eight screens for the mobile application. The first two screens are Login and Sign Up. These screens allow the user to log in and sign up for the application. After the user logs in, they can explore the product offers in the Home Screen, Search, and Product List. Once they have added different items to the cart, they can navigate to the Cart screen and proceed to checkout in the Checkout screen. The product lists and search results are retrieved from the backend system. Similarly, recommendations are also generated and returned back to Mobile Application for display using the Client-Server model mentioned in Section 3.3.2.

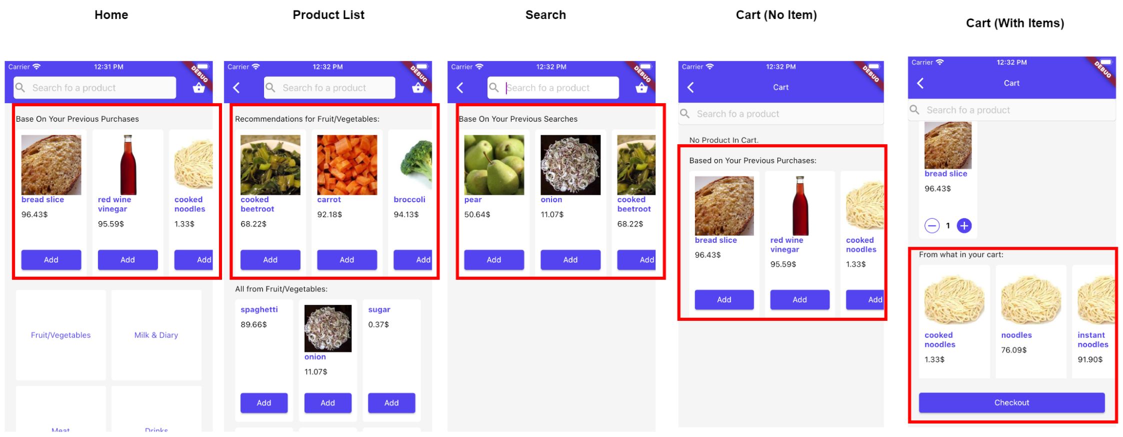

Figure 12: Recommendations Shown in Mobile Application

We show recommendations to the users on the mobile application in four different areas. First, we display *Base on Your Previous Purchases* recommendation on the Home Screen. This is to provide the user quick access to products that are based on their purchase pattern. Second, we show *Base on Your Previous Searches* recommendation on the Search screen. We record what the user has searched for in the past and use that information to find similar products to the ones they have searched for. Third, we should *Similar products recommendation* in the Product List screen. This provides the user with similar items that might not be in the same category but has similar characteristics. It should provide users with more diverse access to products. Finally, we also show *Base on what's in your cart* and *Frequently bought together* recommendations in the Cart screen. This gives users quick access to products that they may forget to add to their cart before checkout.

3.3.4 Backend System

Route	Method	Description
/auth/login	POST	log in with email and password.
/auth/signup	POST	Sign up the user with email and password.
/users	GET	Get all users
/users/user_id	GET	Get a user
/users/create	POST	Create a user
/users/user_id	PATCH	Update a user
/users/user_id	DELETE	Delete a user
/products	GET	Get all products
/products/product_id	GET	Get a product
/products/create	POST	Create a product
/products/product_id	PATCH	Update a product
/products/product_id	DELETE	Delete a product
/cart/user_id	GET	Get all items in user's cart
/cart/add	POST	Add an item to a user's cart
/cart/cart_id	DELETE	Delete an item from a user's cart
/interactions	GET	Get all interactions
/interactions/create	POST	Create an interaction
/interactions/interaction_id	PATCH	Update an interaction
/interactions/interaction_id	DELETE	Delete an interaction
/interactions/interaction_id	GET	Get an interaction
/checkout	POST	Process a payment
/frequently_bought_together/product_id	GET	Get frequently bought together products with product_id.
/similar_products/product_id	GET	Get similar products to product_id
/others_also_bought/interaction_id	GET	Get products that others users with similar characteristics also bought

Table 2: API Endpoints

Our backend consists of 7 API groups: (i) Authentication, (ii) Carts, (iii) Interactions, (iv) Payment, (v) Product, (vi) Recommendations, and (vii) Users. In Authentication, we developed two API for login and signup. The Flutter Mobile application calls this API to perform authentication for new and existing users. To perform recommendations, we provide the Mobile Application access to three main recommendations routes. First, the *frequently_bought_together* route can be used to provide recommendations for frequently bought together items in the cart. Second, the *similar_product* and *others_also_bought* routes is provided for usage with *Based on your previous* and *Similar Products type recommendations* type recommendations. Table 2 outlines all of the API ends points we have developed for the backend system along with a short description.

3.3.5 Overview of Code Listing

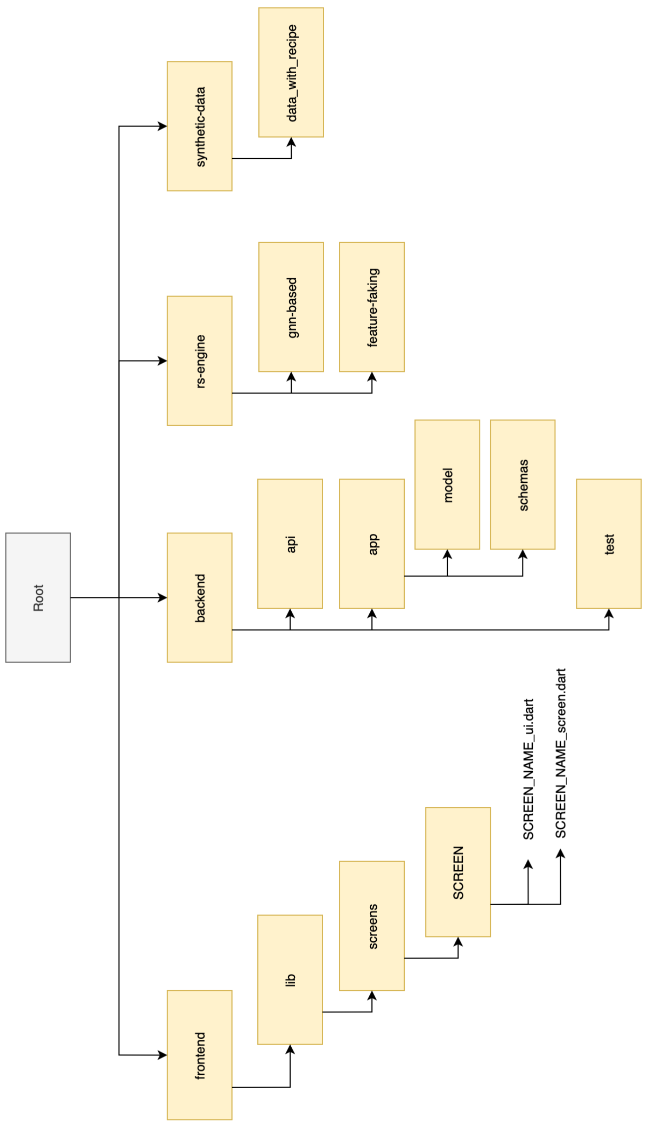

Figure 13: Overview of Code Listing

Our code base is divided into 4 major components: (i) *frontend*, (ii) *backend*, (iii) *rs-engine*, and (iv) *synthetic-data*. We divide them in this way to create a separation of concerns between all the major components of the project.

- *frontend* consists of code base for the Flutter Mobile Application project. We follow a popular coding pattern called Smart/Dump components in frontend. As a result of each the screens we code consists of two files *SCREEN_NAME_ui.dart* and *SCREEN_NAME_screen.dart*.
  SCREEN_NAME_ui.dart contains all layout based code such as button placements and coloring while *SCREEN_NAME_ui.dart* consists of logic based code like button presses and API calls.
- *backend* houses all of the Flask project for our backend system. Here we seperated our the code containing the API calls from our database models and schemas definitions. This is to ensure scalability because API related code tend to be edited more than database related code.
- *rs-engine* contains all of the code we used for our experimentation with different recommendation approaches. Each approaches contains code for running the training, inferences, and evaluation.
- *synthetic-data* houses our code base for generating the synthetic data discussed in Section 3.1.1. It contains all the code for generating purchase, users, and products data. It also contains the generated data used in this project.

Furthermore, in each component, we have provided README files with instructions for installing the neccessary packages and for running the project.

4 Experimental Setup

This section outlines the experimental setup carried out to trained and test the perform of the different recommendation approaches on the synthetic data. It first discusses the dataset and evaluation metrics. Then, it ends with the discussion of the baselines used and the training configuration of each of the recommendation approaches mentioned in Section 3.1.5

4.1 Dataset and Evaluation

The dataset used for the training of different recommendation approaches are generated using our novel synthetic data generation approach outline in Section 3.1.1. There are three main sets: (i) Users, (ii) Products, and (iii) Interaction (Purchase Data). The interaction file contains 400,000 purchases across 400 customers. A train/test split is created where 80% of a customer purchases is kept for training and the remaining 20% is used for testing. This resulted in a training set that has 800 purchases and a test set that has 200 purchases for each of the 400 customers. Finally, to evaluate the different recommendation approaches, we calculate their performance in terms of Accuracy@10, Precision@10, and AveragePrecision@10 on the test set.

$$\text{Accuracy@}k = \frac{1}{N}\sum_{i=1}^{N}\frac{|\{\text{relevant items retrieved}\} \cap \{\text{top } k \text{ items}\}|}{\min(k, K)} \tag{5}$$

$$\text{Precision@}k = \frac{1}{N}\sum_{i=1}^{N}\frac{|\{\text{relevant items retrieved}\} \cap \{\text{top } k \text{ items}\}|}{k} \tag{6}$$

$$\text{AveragePrecision@}k = \frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{j=1}^{k} P(j)\cdot \text{rel}(j)}{K} \tag{7}$$

where

- N: the number of queries.
- k: the number of items to retrieve.
- K: the total number of relevant items.
- $P(j)$: the precision at position j
- $\text{rel}(j)$: an indicator variable that equals 1 if the item at position j is relevant and 0 otherwise.

4.2 Baselines

We have chosen three baselines to compare with our recommendation approaches. First, for content-based type recommendations, we trained them on traditional values as a baseline. Second, for collaborative filtering, we used an open-source implementation from Implicit **frederickson˙implicit˙2023**. Third, to ensure that our algorithm performs better than random, we developed a module to produce recommendations by randomly selecting products.

4.3 Feature Faker

The algorithm is trained by running through all iterations once. Having it run more than once may impact the model's performance to favour a certain item over others. Following that one hundred unknown features are selected through trial and error. In order to find out the optimal number of features, the model requires a search algorithm to find optimal product connections with features. One hundred features were chosen because the model ended up outputting a satisfying performance.

4.4 Graph Based Recommendation

We trained our GNN model for 200 epochs with a batch size of 128. Furthermore, we used the Adam optimiser **kingma˙adam˙2014** with a learning rate of 0.001. For a GNN to learn to predict the existence of an edge, we need to introduce false/negative edges during training. Therefore, we add these negative edges during the data splitting process with a ratio of 2 to 1 of the positive edges.

4.5 Auto-Encoder Recommendation

With a 128-batch size, we trained our Auto-encoder model for 100 epochs. Additionally, we applied the Adam optimiser **kingma2014adam** with a 0.001 learning rate.We have also use Early stopping, which is a neural network approach that helps with generalization and avoids overfitting **girosi1995regularization**. During training, the network's performance is monitored against a validation set, and training is terminated when the validation error starts to rise or stops decreasing. This is done to avoid the network getting highly specialized in the training material and unduly learning it, which could result in subpar performance on new, unforeseen data. However, the performance of the network during training is monitored using the historical technique. This is crucial since it enables you to keep track of the network's development and identify any potential problems.

5 Testing, Result and Discussion

This section will cover pros and cons of synthetic data generation. The performance of the Feature Faker, Modified graph sage, auto-encoder and baseline approaches on that data. Then the performance of the algorithms on the app. Followed by testing methods used for front-end and backend. The final section will discuss the results of the carried out experiments.

5.1 Synthetic Data Generation

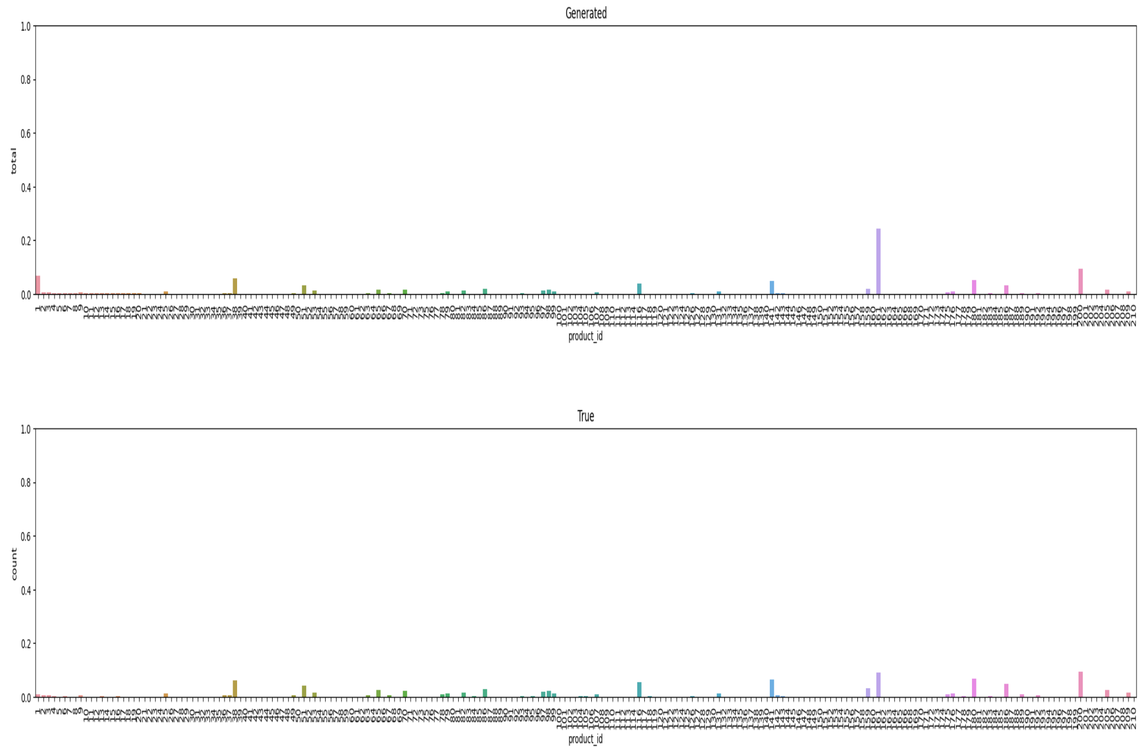

Figure 14: Generated (Top) vs Real (Bottom) Purchase Data

Based on the methodology outline in Section 3.1.1, our output shows that the distribution of the products bought in the generated set matches the data of the actual distribution. However, there are two notable characteristics. First, our approach resulted in products with low bin counts being decreased further relative to the most frequent product. We believe that this is caused by two factors: (i) Small bin count products get chosen less as starting products. (ii) The 4Gram model used also resulted in fewer small bin-count products being suggested as the next product purchased. Second, our approach resulted in a very high bin count for beers. Investigation shows that the reason behind this is that during the matching of product names between the open-source data and Nosh products, the majority of soda-based products were matched to beer. This makes sense because there are fewer unique products in the Nosh products list compared to the open-source data. More specially, there are only two types of drinks in the Nosh Product data (milk and beer). We believe that increasing Nosh product offers would yield better base distributions that should result in better synthetic data being generated.

5.2 Recommendation Approaches

Model	**Accuracy@k**	**Precision@k**	**AveragePrecision@k**
Random	0.0505	0.0505	0.01583
Implicit Alternating Squares	0.0045	0.0045	0.0014
Feature Faker	**0.484**	**0.484**	**0.2539**
Modified GraphSage	0.008	0.0082	0.0018
Euclidean Distance (Baseline)	0.006	0.006	0.003
Auto-Encoder	0.004	0.004	0.002

Table 3: Evaluation Results of Recommendation Approaches

Table 3 displays three interesting characteristics. Firstly, all models achieve scores below 0.5. We believe that this is due to the nature of training and testing on synthetically generated data. Even though our synthetic data matches the distribution of the actual buying patterns of an open-source dataset, the resulting data may have more randomness compared to real purchase patterns. Hence, it is harder for our chosen models to learn. Second, randomly choosing products produces higher results than the Graph Base and Implicit approaches. It may be the result of our NGram implementation as part of the synthetic data generation. We introduce a randomness factor when picking the top most likely to be the output of our 4Gram. Therefore, randomly selecting products does match a small fraction of our generated purchase data. It is probable that results would be different if the experiments would be carried out on a real data set and not a synthetic one. Finally, all algorithms outperformed Implicit. Consequently, the results of the recommendation algorithms are considered to be successful since Implicit is our baseline approach.

5.3 Testing Methodology

This section outlines the testing methodology adopted for the frontend (Flutter) and the backend (Flask). This section has already been discussed in our Software Requirement Specification document. However, we reiterate it here for quick reference. We employ three different strategies. First, in both the frontend and backend, we have aimed to do unit testing. This is to ensure that each unit of our system performs as expected. In the frontend case, these unit test is done on individual UI components while in the backend case it is be done for each individual API. After the development of the backend and frontend has been completed, we performed integration testing. In this stage, we did black box testing by going through a checklist of the expect behaviour of the system when we integrated the frontend with the API. Finally, we also performed Acceptance testing after the completion integration testing. In this final stage, validate the mobile application and backend systems with Nosh Technology to ensure that it meets all the major requirements.

5.4 Test Results

Due to time constraints we are unable to cover our code base with unit tests. However, we have setup code structure for the testing to proceed for the Flask backend. We have installed PyTest and setup code that would allow for the testing of the individual APIs. Despite this, we did performed integration testing and acceptance testing successfully. We have demonstrated to Nosh Technology the demo application with a complete API integration which have been accepted as matching the requirements laid out in our Requirement Specification Document. The backend testing was carried out on a program called ”Postman”, which simulates API calls.

5.5 Discussion

This section summarises and analyses the results and findings of Recommendation approaches, front-end system, and back-end systems.
Recommendation Approaches: Two types of experiments were carried out to test the recommendation models. The first experiment contains the synthetic data. Feature Faker performed extremely well in experiment one. Surprisingly Followed by random baseline approach. The other recommendation models have outperformed the main baseline approach - Implicit. Which is a satisfying performance for experiment one. The second experiment was carried out on the app itself. Rather than testing the accuracy, we tested how believable and natural the recommendations feel. Where we have decided that GNN was the best one. Therefore it was the one implemented into a final product.

Front-end: The front-end system was tested if it complies with the system requirements stated in the software requirements document. A unit testing was done to make sure that all the mobile app screens and buttons are accessible by users.
Back-end: Regardless of little testing done on the backend, it ended up working reliably. There were no unit testing implemented, most of the testing was done with "Postman" app. Later on, more test were done after the front end and backend was merged in order to check for responsiveness. In the end tests are a useful tool to find bugs, despite that it has no other important influence on the backend and could always be easily implemented in the future.

5.6 Recommendation Approach Selection for Nosh Technology

Based on these findings we have chosen to deploy the GNN based method as our main recommendation engine for usage in the Nosh Technology mobile application. There are two reason why we have made this choice despite its lower accuracy compared to Feature Faker. First, the GNN outputs users and products embeddings that can be used for both Collaborative and Content Based type recommendations. This is useful because it limits the number of models Nosh Technology have to maintain. Second, the GNN based approach can handle the addition of new users and product without the need to retrain. This is useful because as Nosh Technology on board more users or adding more product offers, retrain a recommendation engine becomes more costly. Hence, using the GNN saves the company future training resources.

6 Conclusion

In conclusion, we have developed a complete AI-based recommendation for food purchases in a mobile application in collaboration with Nosh Technology. To realise this goal, we have developed a novel approach for synthetic data generation for food purchase, a novel recommendation approach called Feature Faker, and a GNN recommendation method based on GraphSage **Hamilton.etal2017**. We then developed a mobile application with Flutter and a backend system in Flask to serve the recommendations to Nosh Technology's users.

While the system might be completed, there are two possible directions for future work. From the recommendation engine side, we believe that a future examination of GNN-based methods can yield improvements in terms of accuracy. Furthermore, a different open-source dataset can be experimented with to generate synthetic data with our method that can yield better results for the different approaches. From the mobile application and backend system side, we believe that one potential improvement is adding a scheduler to the backend to update users and product embedding on a regular basis. This process should ensure that the users and product embeddings stay updated with new interactions added to the system.

Overall, we have completed the core components listed in the requirements. Although there can be further improvements, they are out of the scope of the development mentioned in the Requirement Specification Document.