Menu Engineering Supercharged

MDS Capstone Final Report

Hankun Xiao, Yasmin Hassan, Jessie Zhang, Zhiwei Zhang

Table of contents

1 Executive Summary

This project supports our partner, "heymate!"(hereafter referred to as *Heymate*), in delivering data-driven menu insights to their restaurant clients. Using 3 million popularity records and large language models(LLMs), we developed a structured data cleaning pipeline and generated a weighted scoring mechanism to identify top-performing dishes. The final product includes a scalable recommendation system that suggests the most popular menu items based on restaurant type, enabling merchants to optimize their offerings with minimal technical effort.

2 Introduction

Our capstone partner, Heymate, offers an all-in-one business management platform, primarily serving restaurant clients. However, many of these restaurant owners struggle to design menus that align with customer preferences and evolving market trends due to limited access to broader market data. Our project aims to bridge that gap by transforming publicly available menu data into structured insights that support data-informed menu design, enabling the partner to deliver greater value to its clients through market-driven recommendations.

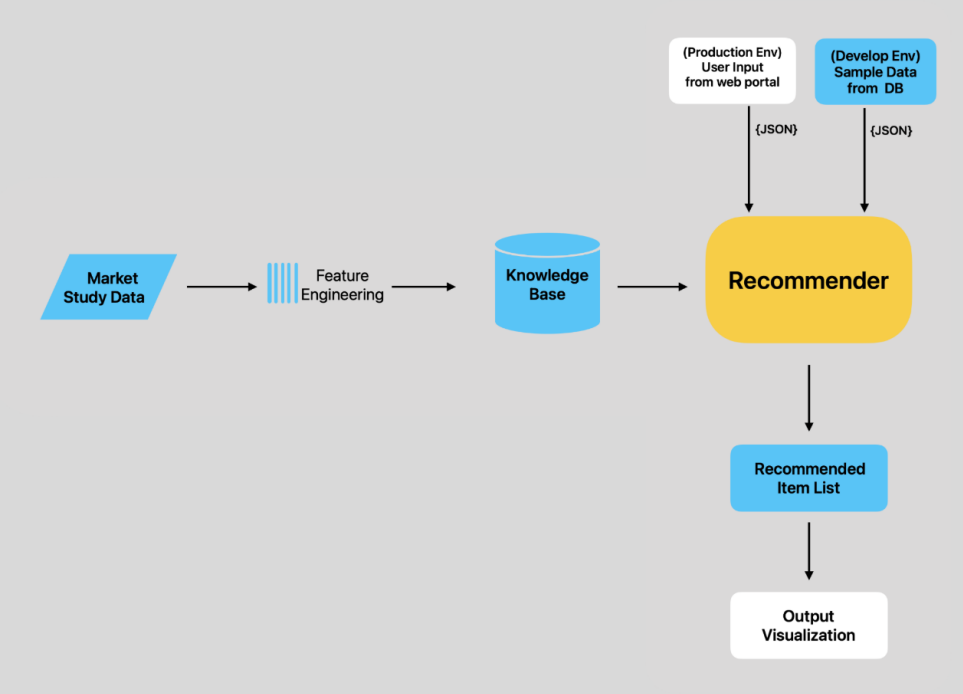
Our initial resource was internal database provided by Heymate, containing thousands of raw menu records from their clients. However, the internal data alone was insufficient to support robust recommendation modeling due to limited scale of menu and lack of trending information. Specifically, the internal dataset only includes around 300 restaurants, which limits its representativeness and diversity for building a generalizable recommendation system. To address this, we explored various open-source datasets, and found a menu dataset containing millions of menu records along with associated popularity metrics on Uber Eats (Sakib (2023)), making it a strong data source in terms of scale, quality, and coverage.

However, messy data, large-scale datasets, and customized menu item names posed significant challenges to the construction of our recommendation system. Therefore, our scientific objective was to design an end-to-end pipeline that leverages diverse data science techniques to build a robust and scalable recommendation system, powered by a popularity-based knowledge base and capable of handling multi-type inputs.

3 Data Pipeline Framework

Based on the project objective, we developed the following modular data pipeline framework: (Figure 1)

Figure 1: Data Pipeline Framework - The core recommender supported by two pipelines: one for modeling from market data, and another for generating recommendations



3.1 Model Input Pipeline (Horizontal Workflow)

1. Data Ingestion: As mentioned in the Introduction, we used an Uber Eats dataset from Kaggle that covers diverse restaurant types, menu items, and descriptions.
2. Feature Engineering: As the raw dataset was quite messy, it needed to undergo a series of preprocessing and standardization processes so that we can use them for further popularity score calculation.
3. Knowledge Base Construction: Under suggestion and domain knowledge from the project partner,we built a weighted scoring knowledge base on various popularity metrics. This serves as the foundation for capturing market trends and powering recommendations.

3.2 Testing and Production Pipeline (Vertical Workflow)

1. Recommender Input: The recommender system can process JSON inputs from two sources, including client restaurant type from Heymate internal database, and user input of restaurant types via the web portal for real-time recommendations.
2. Recommendation Module: Based on the specified restaurant type, the engine searches the knowledge base, ranks menu items using scoring logic, and generates a tailored Recommended Item List.
3. Output Visualization: Final recommendations are displayed through a user-friendly dashboard, enabling clear interpretation and easy action on the results.

4 Transition from Framework to Data Product

Having established our framework, we shifted our focus to building a scalable data product. At its core is a recommendation engine that suggests top-performing dishes by leveraging aggregated popularity patterns from similar restaurant types.

4.1 Data Wrangling Module

4.1.1 Exploratory Data Analysis

To support the development of a robust recommendation engine, we first conducted exploratory data analysis (EDA) to assess data quality and structure. The insights suggest the need to perform wrangling tasks for a clean, consistent, and high-quality model input dataset.

1. **Data Quality and Inconsistencies:** The raw Uber Eats data contains more than 5 million menu records, but had quality issues (shown in Table 1), including missing metrics and inconsistent formatting. Therefore, we implemented preprocessing logic to remove all these incomplete, duplicated, or other invalid menu records, which results in around 3.2 million records for next step.

Table 1: Null Entries in Uber Eats dataset (Model Input Data)

column_name	null_count
id	0
restaurant_name	0
score	1958476
ratings	1958476
restaurant_type	2499
full_address	33745
menu_category	160
menu_name	164
menu_item_description	1452305
price	160

2. **Large-Scale Data Processing:** Even after preprocessing, the dataset still contained over **3 million** records. Details of our deployment approach are discussed in the Data Science Methods section.

3. **Menu Naming Variability:** Inconsistent naming of similar dishes (e.g., various forms of "seafood fried rice") made it difficult to compare and group items. We address this challenge in detail in the Data Cleaning Module.

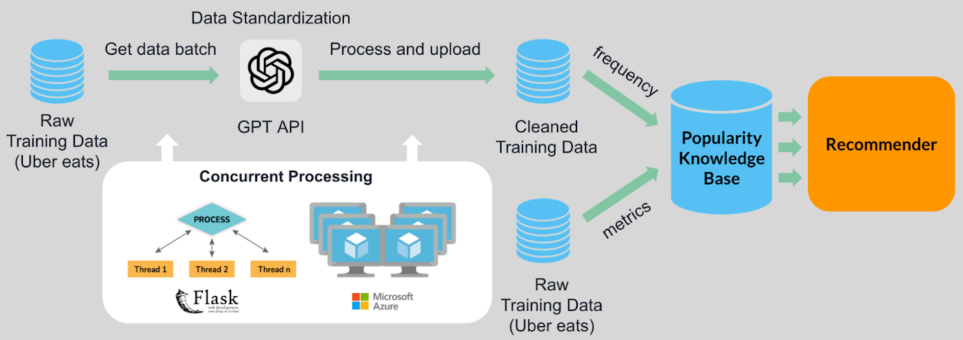
Figure 2: Seafood Fried Rice Spelling Variation

Product Name Variation	Restaurant Name
Seafood Fried Rice 魚蝦蟹炒飯	Fisherman's Terrace
G25. Sauteed Seafood Fried Rice 避風塘海鮮炒飯	Yue Ting Seafood Restaurant
149. Fook Chow Seafood Fried Rice福州海鮮炒飯	Double Double Restaurant & Wonton Ltd.
160. XO Sauce W/ Seafood Fried Rice XO 醬海鮮炒飯	Double Double Restaurant & Wonton Ltd.
161. Dried Scallop & Seafood Fried Rice瑤柱海鮮炒飯	Double Double Restaurant & Wonton Ltd.
Seafood Fried Rice	Nishiki Sushi Bar
Assorted Seafood Fried Rice	So Good Restaurant
Seafood Fried Rice 鴻羊炒飯	Fortune Lamb Dining
Seafood Fried Rice牛油海鮮炒飯	Richmond
Seafood Fried Rice 海鮮炒飯	Lougheed
Fu-Chow Style Seafood Fried Rice 福州炒飯 (Topped ...	Lougheed
Foo Chow Style Seafood Fried Rice / 福州炒飯	Pelican Seafood Restaurant
Seafood Fried Rice with Pineapple / 海鮮炒飯	Pelican Seafood Restaurant
Deluxe Seafood Fried Rice / 有米海鮮炒飯	Pelican Seafood Restaurant
35. Seafood Fried Rice	Summer House
Seafood Fried Rice	Valendine
43.芝士白汁焗海鮮飯或意麵 Baked Creamy Seafood Fried Rice...	Neptune Chinese Kitchen (UBC)
921. 海鮮瑤柱炒飯 Dried Scallop & Mixed Seafood Fried...	Neptune Chinese Kitchen (UBC)
926. 海鮮粒炒飯 Diced Mixed Seafood Fried Rice	Neptune Chinese Kitchen (UBC)
3. Seafood Fried Rice Served In Whole Pineappl...	Grand Crystal Seafood Restaurant
Seafood Fried Rice 海鮮炒飯	Pinyuexuan Seafood Restaurant
931. 海鮮炒飯 Seafood Fried Rice	Lucky Fortune Seafood Restaurant
102. Fu-Chow Style Seafood Fried Rice 福州炒飯	Lucky Fortune Seafood Restaurant

4.1.2 Solution Pipeline

All wrangling tasks were encapsulated in modular Python scripts and integrated into a unified data pipeline (Figure 3). This pipeline includes:

Figure 3: Solution Pipeline (Model Training)



1. **Data Cleaning**: To handle naming inconsistencies across restaurants, we used an LLM-powered semantic standardization pipeline to group similar items under unified labels (details in Data Cleaning Module).
2. **Cleaning Deployment**: The cleaning pipeline runs in batches via a Flask-based batch-processing mechanism for parallel execution. Each batch of raw data was processed and uploaded to generate a cleaned dataset.
3. **Scoring and Knowledge Base**: Cleaned items are merged with key popularity metrics to build an aggregated table for popularity score computation (details in Recommendation Algorithm).

For the model testing dataset (Figure 5), the similar logic is applied here as Heymate's internal database also exbihits some extent of quality issues (shown in Figure 4). The standardized input will pass through the recommender system to generate top-rated dish recommendations.

Figure 4: Null Entries in internal dataset (Model Testing Data)

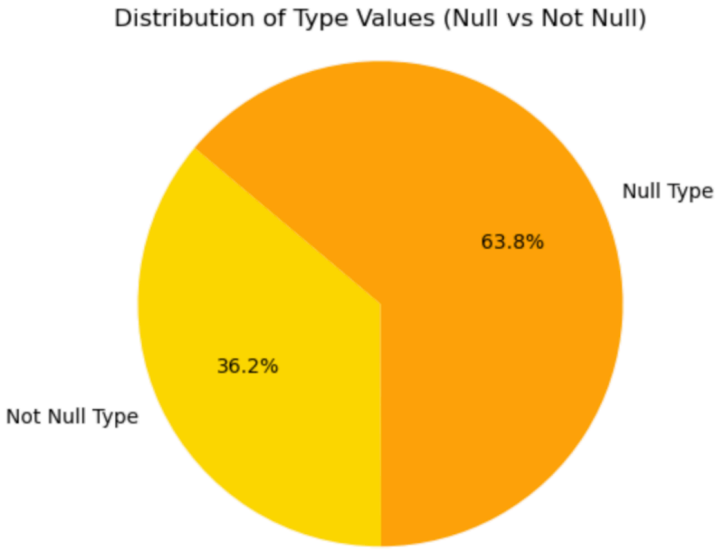
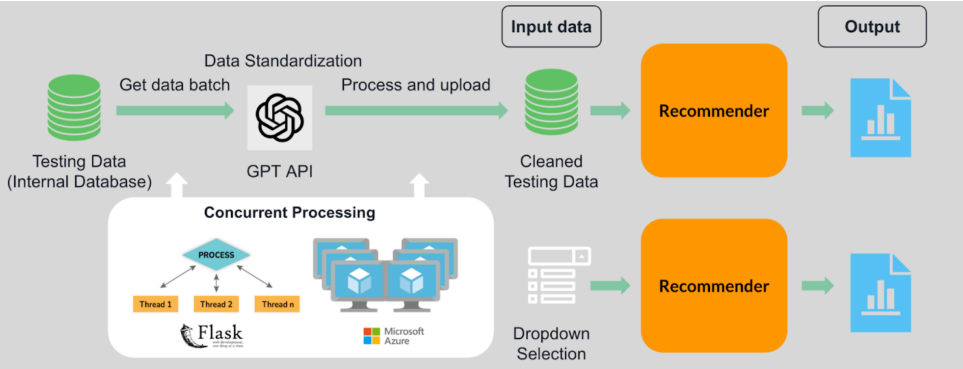


Figure 5: Solution Pipeline (Model Testing)



4.2 Data Cleaning Module

After reviewing the overall pipeline, we now take a closer look at the data cleaning component, which plays a key role in addressing inconsistencies in menu dish names and enhancing the quality of downstream recommendations. Specifically, we want to address following issues:

- Mixed languages: Many item names contain both English and non-English text (Figure 6).
- Inconsistent restaurant types: Broad categories (e.g. "American") are mixed with specific terms (e.g. "wings", "sandwich"), making aggregation difficult.
- Combo indicators: Expressions such as "Combo", "Set of Two", or "Family Meal" are used inconsistently, complicating identification.
- Quantity terms: Words like "6 Pc" add additional noise and make parsing less accurate.

Figure 6: Cleaning Challenge Example

	menu_item_name
0	Beef Teriyaki Set
1	Sambusa I ሳምቡሳ 3pc
2	煤氣爐 Cassette Gas Cooker

Due to limited time, we decided to utilize ChatGPT API as it can achieve semantic inference, format alignment, as well as language processing more efficiently while maintaining quality. We developed prompts that includes two main parts:

1. System prompt: Defines the input and output schema as well as cleaning rules, such as how to identify the core dish name, extract up to five descriptors, determine whether an item is a combo, and standardize the restaurant type.
2. User prompt: Feeds batches of raw menu rows into the model in a list of dictionary format.

The ChatGPT model returns structured output for each menu item, including (Figure 7):

- `dish_base`: core identity (e.g. "fried rice").
- `dish_flavor`: tags like cooking method or toppings (e.g. ["chicken"]).
- `is_combo`: boolean indicating whether the item is a combo.
- `restaurant_type_std`: standardized restaurant type aligned with Google Maps Food and Drinks category.

To ensure extraction quality and reliability, we iteratively refined our prompt engineering strategy:

Figure 7: Cleaned Menu Item Example

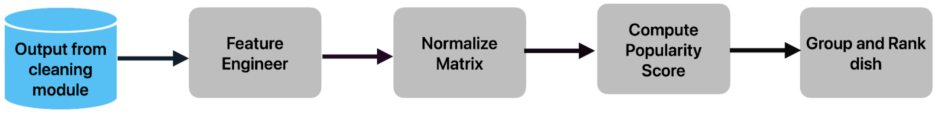
	dish_base	dish_flavor	is_combo	restaurant_type_std
0	rice	pork chop, gravy	True	breakfast restaurant
1	sausage	egg	True	breakfast restaurant
2	breakfast sandwich	bacon, egg, cheese	False	breakfast restaurant
3	burger	double cheese	False	breakfast restaurant
4	bacon	egg	True	breakfast restaurant

- Required vs. optional fields: We clearly specified which fields (e.g. item name) must be present, and which are optional (e.g. menu description). When optional fields were missing, the model was instructed to infer based on other inputs.
- Formatting rules: We enforced strict formatting in the output, including lowercase, singular, American English spellings, to ensure consistency.
- Controlled restaurant type output: We constrained the model to select from a fixed list of restaurant types aligned with Google Maps Food and Drinks categories.
- Combo identification: We embedded recognition logic for different combo indicators, such as "set of", "combo", or "family meal".
- Prompt length optimization: Through iterative testing, we shortened prompt size while maintaining output quality, helping reduce API costs.
- Row indexing: Each row in the batch was assigned a unique index to link the model's structured output back to other features like rating or score.

With this module, we were able to clean inconsistent menu data into a structured format suitable for downstream analysis and recommendation.

4.3 Recommendations Algorithm

Figure 8: Recommendations Workflow



4.3.1 Feature Integration

Once the data cleaning module produces cleaned menu data (stored in `cleaned_menu_mds`), we enrich it by joining with the Uber Eats metadata table (`Restaurants_mds`) to reintroduce three key popularity indicators that had been excluded during the cleaning phase:

- Number of Ratings: how many users rated the dish
- Average Rating: the restaurant average rating score given by users
- Item Frequency: how often a dish appears, counting occurrences grouped by dish base, dish flavor, and restaurant type

To ensure that combination meals don't distort the popularity signals, we filter out all entries flagged as combos (`is_combo == True`). The resulting dataset serves as the foundation for our **core knowledge base**.

4.4 Compute Popularity Score

We applied **MinMaxScaler** to rescale frequency, rating count, and rating score to a [0, 1] range. We then calculate a weighted popularity score for each dish as follows:

```
popularity_score = (0.2 * freq_scaled + 0.6 * rating_scaled + 0.2 * score_scale)
```

Weights were selected based on testing and partner input, prioritizing review count as a signal of engagement, an approach supported by industry practices/research in similar recommendation systems (Chitalia (2023) Al-Rubaye and Sukthankar (2020)).

- **60% rating count: (`rating_scaled`)** Measures how many people have reviewed; we viewed it as the strongest proxy of broad customer engagement.
- **20% average score: (`score_scaled`)** captures perceived quality but is often biased by low volume.
- **20% frequency: (`freq_scaled`)** reflects widespread presence on menus but doesn't always indicate desirability.

All computed scores are stored in a SQL table: `cleaned_menu_with_popularity` This becomes the knowledge base table used for filtering and recommendations.

4.5 Group and Rank Dishes (Filtering & Output Logic)

At recommendation time, the system:

1. Accepts input from a restaurant partner (up to 3 restaurant types)
2. Filters the knowledge base (`cleaned_menu_with_popularity`) using:
    - Exact matches on `restaurant_type_std`
    - Filters out duplicate dishes (e.g. same base/flavor combo for the same type)
3. For multi-type requests, the popularity scores are averaged across the selected types
4. Dishes are then ranked based on their average popularity score

Finally, the top N dishes (configurable) are returned as output.

5 Example Use Case: Recommending for a Pizza Restaurant

To demonstrate how the recommendation engine works in practice, we include a use case where a partner restaurant is identified as a **"pizza restaurant"**. The system filters all dishes from our cleaned Uber Eats knowledge base that match this restaurant type and scores each based on its normalized rating score, frequency, and rating count. The weighted score is then used to rank the items, returning the top 10 most relevant dishes.

As shown in the figure below Figure 9, the top items for a pizza restaurant include classics like pepperoni pizza, meat lover's pizza, and spaghetti. Less expected items like Papadia also rank highly due to strong metric combinations.

Figure 9: Recommendations For a Pizza Restaurant”

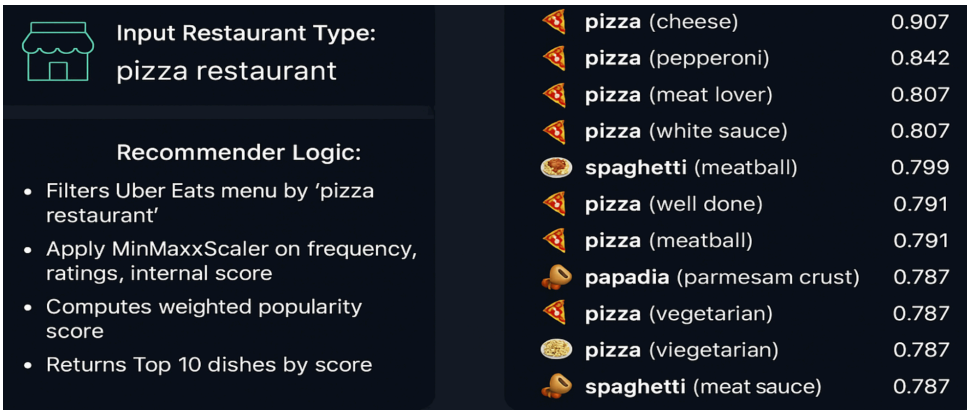
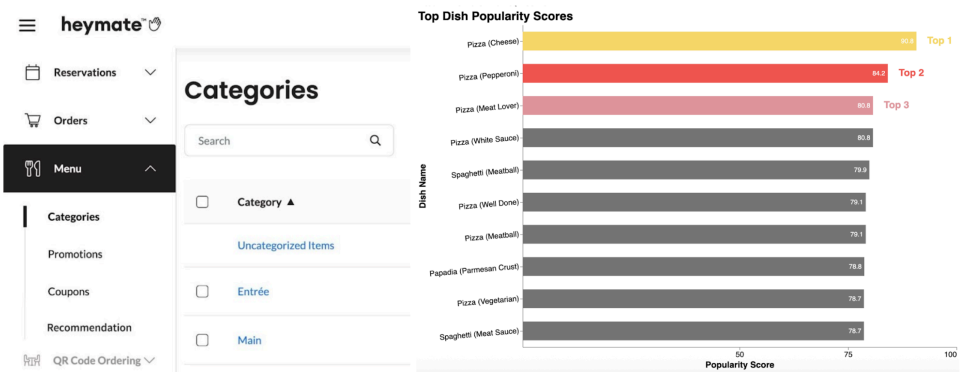


Figure 10: Visualization Demo”



5.1 Visualization Demo: Surfacing Actionable Recommendations

To make our recommendation results interpretable and actionable, we built a two-part visualization demo:

- **CRM Integration Mockup (Left of Figure 10):** A conceptual UI showing how Heymate's CRM can embed our recommender. This allows managers to access top dish insights directly from their dashboard. The Heymate engineering team plans to build this interface in future development.
- **Altair Bar Chart (Right of Figure 10):** A visual of top dishes sorted by popularity score. Features include:
  - Heymate brand colors for top 3 dishes
  - Percentage labels and hover tooltips for easy exploration
  - Combined dish base and flavor names for clarity

This design ensures our scoring pipeline results are both explainable and ready for integration.

6 How to Use the Data Product

We designed this product to help Heymate and its clients make informed menu decisions.

6.1 Internal Use

This internal setup enables iterative refinement of the pipeline:

- Data Updates: Heymate staff can upload new datasets to evaluate updates and improvements.
- Testing & QA: Input client restaurant types to verify dish name cleaning and recommendation logic.

6.2 Client Deployment

To support merchants in aligning with current market trends, this tool enables data-driven menu decision-making:

- Restaurant Type Input: Clients select a restaurant type (e.g., "pizza restaurant").
- Recommended Output: The system returns a list of top recommended dishes based on aggregated popularity metrics from similar restaurants.

7 Data Science Methods

To build a scalable pipeline supporting both internal testing and client deployment, we applied three key data science techniques: **LLM Integration**, **Distributed Deployment**, and **Materialized View**. In the following sections, we describe how each method was applied, along with its limitations and alternative approaches considered.

7.1 LLM Integration

We used large language models (LLMs) to clean and standardize messy menu data. As discussed in the Data Cleaning Module, the raw data was not available for direct use in our recommendation system. The LLM extracted core dish information (base, flavor, combo status, restaurant type) with high accuracy, even without full descriptions.

7.1.1 Limitation

This approach requires payment for API usage. After evaluating trade-offs between computational power and cost, we chose to use the **ChatGPT-4o mini** model.

7.1.2 Alternative Methods Considered

- Regular Expressions: Effective for extracting structured patterns, but not feasible here due to inconsistent formatting and multilingual input.
- Custom Deep Learning Model: While potentially powerful, this would require labelled training data and significant time and computational resources. We don't have such resources within our project scope.

7.2 Distributed Deployment

As mentioned in the Exploratory Data Analysis section, the raw Uber Eats dataset originally contained over 5 million rows. Cleaning this amount of data sequentially would have taken more than 5,000 hours. To speed up the process, we used distributed deployment to process multiple batches at the same time. We deployed 20 workers in parallel, achieving a 20× speedup.

We initially planned to use Azure Functions as suggested in the Developer Guide(Microsoft 2024), to align with our partner's infrastructure. However, due to security configuration issues, we couldn't proceed. As a fallback, we deployed the system locally using Flask.

7.2.1 Limitation

The number of concurrent worker instances is limited by the ChatGPT API rate limit.

We explored other cloud deployment options, including Amazon EC2 and Google Cloud Functions. However, since our partner uses Microsoft Azure, we decided to stay within that ecosystem. In the future, their engineering team plans to migrate our local deployment to Azure Functions once the necessary security configurations are in place.

7.3 Materialized View

To speed up recommendations, we implemented a Materialized View, a database optimization technique that stores query results as a physical table, reducing query time from 6 minutes to 3 seconds and making the system suitable for integration into Heymate's CRM.

7.3.1 Limitations

- Storage Cost: Adds minor overhead due to caching, but manageable.
- Maintenance: Requires refresh after each data update. We automated refreshment after each successful data ingestion.

8 Justification Over Other Products/Interfaces

There are existing solutions that rely entirely on LLMs to build AI agents for restaurant recommendations. In these systems, user inputs are translated into prompts and sent to an LLM, which generates recommendations based on its internal knowledge.

However, this approach has several clear limitations:

- LLMs are transformer-based models and are not well-suited for handling structured logic or computations involving large-scale tabular data.
- Interpretability is low: these systems often function as black boxes, making it difficult to understand or explain how the recommendations are generated.
- Lack of real market data: These models typically do not incorporate up-to-date or domain-specific datasets. In contrast, our system is built on a dataset of over 3 million real menu records, providing a much more grounded and data-driven foundation for recommendations.

9 Conclusions and Recommendations

9.1 What Problem Were We Solving?

Heymate seeks to empower its restaurant partners by offering data-driven menu recommendations that encourage customer return visits. However, partners often lack clear insights into which menu items perform well across the market and why. Our project aimed to close this gap by designing a popularity-based recommendation system that scores dishes based on the number of ratings, average rating scores, and frequency across menus, offering partners a transparent foundation for data-backed decision-making.

9.2 How Does Our Solution Address It?

We developed a full-stack pipeline that:

- Cleans and standardizes restaurant menu data using LLMs,
- Joins the cleaned internal menu with Uber Eats data to enrich it with restaurant-level popularity signals,
- Computes a weighted popularity score using MinMaxScaler across three metrics,
- Filters and ranks menu items based on restaurant type(s),
- Visualizes the results in an interactive Altair chart for use in Heymate's CRM.

This product functions well as a minimum viable recommendation engine, particularly for restaurant partners without access to historical purchase data. Its transparent logic and simple interface make it accessible for immediate use and experimentation.

9.3 Limitations

While our data product offers a practical solution for transforming raw menu data into structured insights, several limitations remain:

- **Popularity Bias:** The popularity score is based on static metrics (e.g., rating count, average score, frequency) rather than real transaction data, serving only as a proxy for customer preference. Future versions could incorporate features like price or order frequency to improve accuracy.
- **Lack of Temporal Insights:** The dataset lacks timestamped records, limiting our ability to detect seasonal trends or menu evolution over time.
- **Rigid Matching Logic:** Recommendations rely on exact restaurant type matches, limiting support for fuzzy or semantic queries like "comfort food." This reduces flexibility for broader user intents.
- **Limited Evaluation:** Current validation is based on case studies due to a lack of real user feedback. A/B testing or real user feedback would be necessary to confirm the effectiveness.
- **Scalability Issues:** Azure deployment is pending due to resource and security limitations. The pipeline is not yet scalable enough for production use cases.
- **Low Personalization:** The pipeline doesn't support sub-types (e.g., "Szechuan") or consider local trends, pricing, or user preferences, limiting personalization.

9.4 Recommendations for Heymate

To evolve this prototype into a production-ready recommender, we recommend:

- **Temporal Tracking**: Incorporate timestamped data to uncover seasonal patterns and trends.
- **Restaurant Feedback Loop**: Integrate feedback from partner restaurants to refine recommendations over time.
- **Semantic Filtering**: Support flexible, natural-language queries using embeddings or LLM-powered matching.
- **Cloud Deployment**: Deploy to Azure to support scalability and integrate directly with Heymate's CRM system.
- **Subtype Extraction**: Enhance LLM outputs with more detailed tags (e.g. "spicy," "gluten-free," "vegan") for deeper filtering.
- **Evaluation Framework**: Design structured evaluations using click data, client interviews, or business KPIs to measure impact.

Our product lays a foundation for scalable, transparent recommendations. With further iteration, it can evolve into a dynamic and adaptive system that supports Heymate's long-term vision for partner success.

References

Al-Rubaye, S., and G. Sukthankar. 2020. "Popularity-Based Ranking of GitHub Repositories." *IEEE*. https://ieeexplore.ieee.org/document/9458206.

Chitalia, A. 2023. "Yelp Popularity Score Calculator." https://scholarworks.sjsu.edu/etd_projects/1261.

Microsoft. 2024. "Azure Functions Python Developer Guide." https://learn.microsoft.com/en-us/azure/azure-functions/functions-reference-python?tabs=get-started%2Casgi%2Capplication-level&pivots=python-mode-decorators.

Sakib, Ahmed Shahriar. 2023. "Uber Eats USA Restaurants and Menus." https://www.kaggle.com/datasets/ahmedshahriarsakib/uber-eats-usa-restaurants-menus.