Menu Engineering Supercharged

MDS Capstone Final Report

Hankun Xiao, Yasmin Hassan, Jiaxin Zhang, Zhiwei Zhang

Table of contents

1 Executive Summary

This project helps restaurant owners make data-informed menu decisions by turning raw menu text into structured insights. We applied language models and scoring techniques to identify popular items and generate actionable recommendations. The solution is scalable and designed for real-world application.

2 Introduction

Restaurant owners constantly face the challenge of designing menus that align with customer preferences and evolving market trends. However, without access to broader market data, these decisions are often based on intuition or limited local feedback. This project aims to bridge that gap by transforming publicly available menu data into structured insights that can support data-informed menu design.

Our capstone partner heymate! provides an all-in-one business management system, with the majority of its clients being restaurants. By structuring and analyzing large-scale menu data, our project enables the partner to offer greater value to their clients through market-informed recommendations and operational insights.

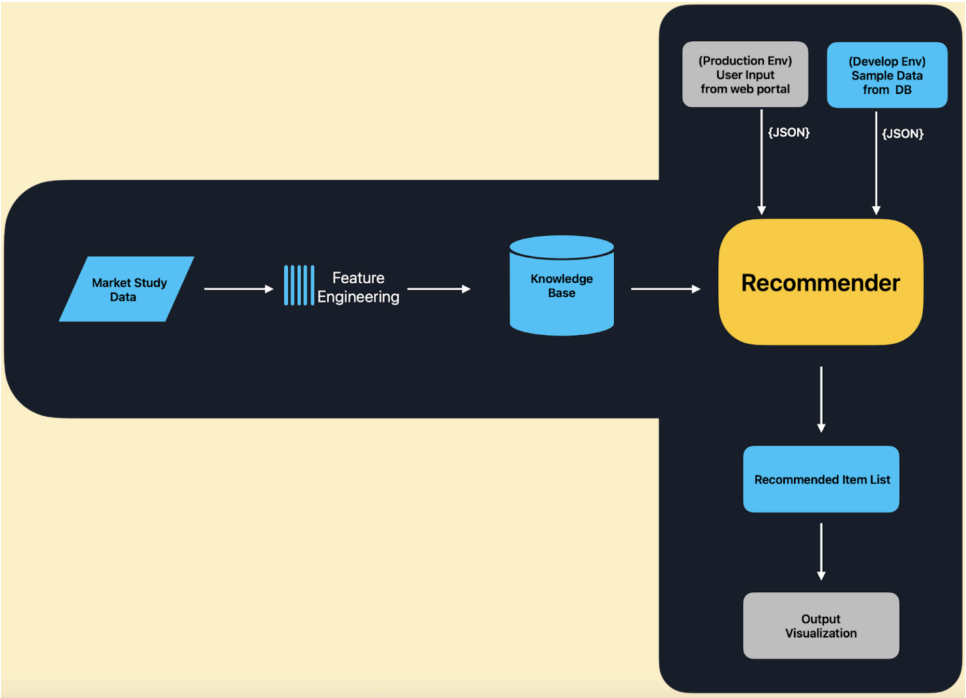
Our initial exploration focused on collecting internal datasets provided by the capstone partner, which contained thousands of raw menu records from their various clients, covering a range of restaurant types. However, the internal data alone was insufficient to support robust recommendation modeling due to its limited scale. To address this, we researched open-source datasets, such as UberEats menus, which included richer item descriptions, categories, and ratings (Sakib (2023)). These datasets helped us understand the variation in menu language, portioning terms, and restaurant classification.

The main objective of our project was to design a structured pipeline that can standardize menu data and generate a scoring mechanism to support recommendation use cases. We built a LLM-based classification module to extract dish base, flavor, combo status, and standardized restaurant types, using carefully designed prompts. These structured outputs were then scored using a popularity metric and visualized for actionable insights.

3 Data Pipeline Overview

The core workflow follows a modular pipeline that integrates data processing, knowledge base construction, and recommendation generation.

Figure 1: Data Pipeline Overview



3.1 Model Input Pipeline (Horizontal Flow)

1. Data Ingestion: We start by collecting raw menu data from both internal and external sources, covering a wide range of cuisines, item types, and descriptions. For this project, we focus on the Uber Eats dataset due to its large volume of menu items and availability of popularity metrics.
2. Feature Engineering: Under suggestion and domain knowledge from the project partner, we identify and select the most informative features from the raw data. The goal is to retain fields that influence customer decisions and remove those that add noise or redundancy.
3. Knowledge Base Construction: We build a knowledge base by scoring each menu item's popularity using appearance frequency, rating counts, and scores. This serves as the foundation for capturing market trends and powering recommendations.

3.2 Testing and Production Pipeline (Vertical Flow)

1. User Input: The recommender system processes JSON inputs from two sources: sampled internal data from the development environment for evaluation and tuning, and live user inputs from the production environment 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 in earlier stages, 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 informed a series of targeted wrangling tasks aimed at producing a clean, consistent, and high-quality model input dataset.

1. **Data Quality and Inconsistencies:** Both the Uber eats and internal datasets presented significant quality issues (shown in Table 1 and Figure 2), including missing values, formatting inconsistencies and test records, posing a major challenge for generating reliable type-based recommendations. To address these, we implemented preprocessing logic that performs null handling, duplicate removal, and validation of key fields before further processing.

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 contained over **3 million** records, making full in-memory processing impractical due to runtime and memory constraints. To ensure scalability, we implemented batch processing by retrieving indexed chunks and deployed it via a Flask framework with multithreaded API calls for concurrent execution.

3. **Menu Description Variability:** The biggest challenge was the inconsistent naming of similar dishes, for example, "seafood fried rice" appeared in various forms and languages (Figure 3). This hindered similarity comparisons and recommendation accuracy. To address this, we used OpenAI's GPT API for semantic standardization, enhancing consistency and model performance.

Figure 2: Null Entries in internal dataset (Model Testing Data)

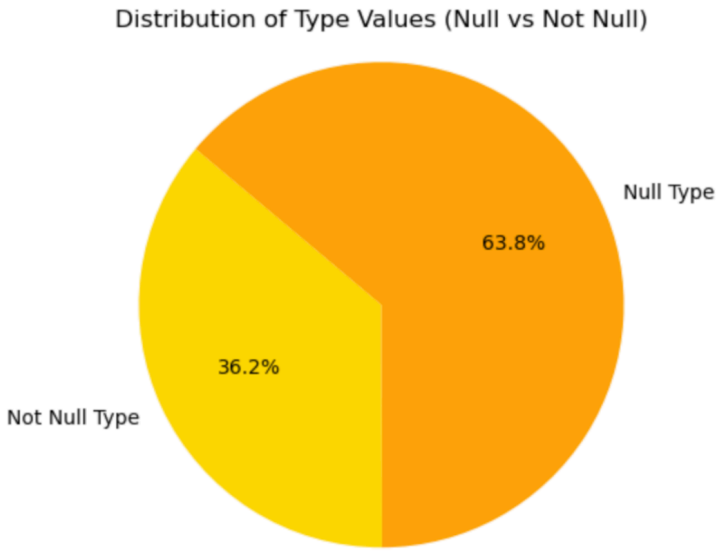


Figure 3: Seafood Fried Rice Spelling Variation

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

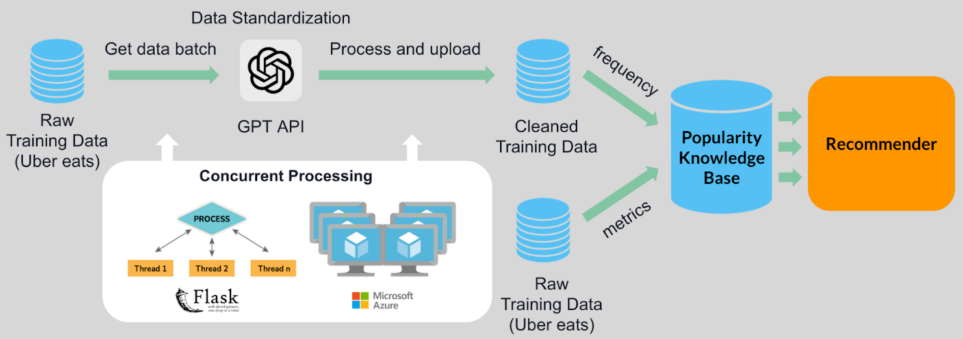
4.1.2 Solution Pipeline

All wrangling tasks were encapsulated in modular Python scripts and integrated into a unified data pipeline. This pipeline:

1. Cleans and preprocesses both internal and Uber eats datasets
2. Applies GPT-based semantic normalization to menu fields
3. Feeds standardized data into the recommendation engine

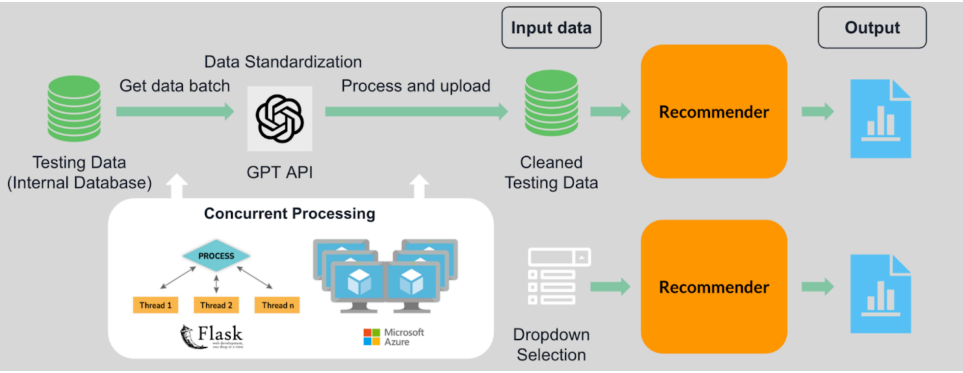
For the model input dataset Figure 4, the processed data is enriched with popularity metrics and serves as the engine's core knowledge base.

Figure 4: Solution Pipeline (Model Training)



For the model testing dataset Figure 5, the same logic is applied to ensure full consistency. The input can be either the cleaned internal restaurant type or the user-selected restaurant type, both of which are passed through the recommender system to generate top-rated dish recommendations.

Figure 5: Solution Pipeline (Model Testing)



After reviewing the overall pipeline, we now take a closer look at the LLM-based cleaning component, which plays a key role in addressing semantic inconsistencies in menu descriptions and enhancing the quality of downstream recommendations.

4.2 Data Cleaning Module

To ensure that meaningful insights can be extracted from raw menu data, we first needed to address a major challenge: data inconsistency.

Both the internal dataset provided by the partner and the Uber Eats dataset exhibit several common challenges, including:

- Mixed languages: Many item names contain both English and non-English text.
- 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.

In addition, the Uber Eats dataset, while richer in features such as ratings, rating counts and descriptions, also suffers from missing values. These missing fields affect the quality of metadata, making it challenging to ensure consistency and completeness during the cleaning process.

Without resolving these issues, it would have been impossible to run any structured analysis across restaurants at scale. To tackle this, we designed a Data Cleaning Module powered by the GPT API. We developed a prompt-engineering pipeline 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 GPT model returns structured output for each menu item, including:

- `dish_base`: core identity (e.g., "fried rice").
- `dish_flavor`: tags like cooking method or toppings (e.g., ["chicken"]).
- `is_combo`: boolean indicating if 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:

- 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.
- Typing 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.

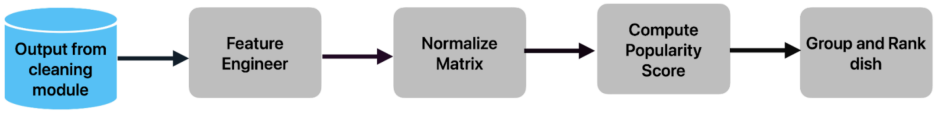
With this module, we were able to clean inconsistent menu data into a structured format suitable for downstream analysis and recommendation.

4.3 Recommendations Algorithm

4.3.1 Feature Engineering

Once the LLM produces a cleaned menu dataset (cleaned_menu_mds), we proceed to feature engineering, enriching this data with relevant popularity metrics. At this stage, we:

Figure 6: Recommendations Workflow



- Join the cleaned Uber Eats menu data with the original Uber Eats metadata tables:
  - `Menu_mds_sorted` (links dish rows to restaurant IDs)
  - `Restaurants_mds` (contains rating counts and scores)
- This join reintroduces three key popularity indicators that had been removed during cleaning:
  - Number of Ratings (ratings)
  - Average Rating (score)
  - Frequency, calculated using a `SQL COUNT (*) OVER (...)` partition grouped by `dish_base`, `dish_flavor`, and `restaurant_type_std` We also:
- Remove entries where `is_combo == True` to avoid noise in popularity estimates
- Eliminate duplicates and ambiguous rows to maintain a clean, reliable feature matrix

The resulting dataset forms the core knowledge base, a structured, popularity-aware version of Uber Eats data, on which all recommendations are based.

4.4 Normalize Matrix

To ensure that the three indicators (rating counts, rating scores, frequency) are **comparable** and **balanced**, we apply **MinMaxScaler** to rescale them to the [0, 1] range. This is important because the raw values of frequency or rating count can be heavily skewed or on different scales. Normalized columns include:

- `freq_scaled`
- `rating_scaled`
- `score_scaled`

This step creates a uniform basis for computing the final popularity score.

4.5 Compute Popularity Score

We calculate a weighted popularity score for each dish as follows:

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

These weights were selected based on exploratory testing and domain intuition from the partner, prioritizing the number of reviews as a strong signal of customer engagement, an approach supported by industry practices/research in similar recommendation systems (Chitalia (2023) Al-Rubaye and Sukthankar (2020))

- **60% rating count:** Measures how many people have reviewed; we viewed it as the strongest proxy of broad customer engagement
- **20% average score:** captures perceived quality but is often biased by low volume

- **20% frequency**:** 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.

4.6 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.

4.6.1 Clarification on Filtering Scope

Currently, our system supports **structured filtering only** specifically, by one to three restaurant_type_std values (e.g., "pizza restaurant", "sandwich shop"). These types are extracted during LLM cleaning and matched exactly during the filtering process. This was inspired by best practices in popularity-based recommenders, which focus on transparency and operational simplicity in early-stage deployment Sreekala (2020). At this stage, we do **not** support:

- Keyword-based filtering (e.g., searching for "pepperoni")
- Semantic/embedding-based matching (e.g., interpreting "comfort food" or "something spicy")
- GPT-powered fuzzy search at recommendation time

This was a deliberate choice to prioritize **transparency, control,** and **execution speed** in our MVP. Additionally, our dataset, though enriched and cleaned, lacked consistent free-text queries or labelled user intent, which made implementing semantic search or GPT-based query interpretation less feasible at this stage.

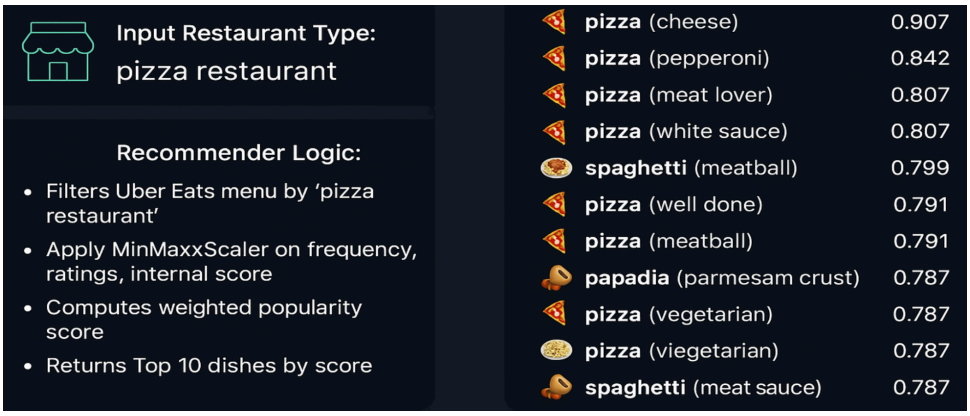
5 Example Use Case: Recommending for a Pizza Restaurant

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, 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 7, the top-ranked items for a pizza restaurant include popular classics such as:

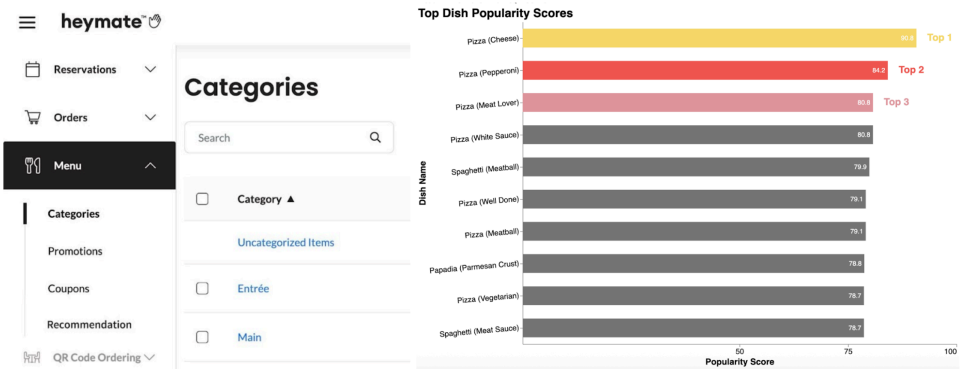
- Pizza (cheese) and Pizza (pepperoni) — widely rated and highly frequent across menus.
- Pizza (meat lover) and Spaghetti (meatball) — dishes with consistent performance.
- Less expected but still relevant items like Papadia (parmesan crust) also appear due to strong metric combinations.

Figure 7: Recommendations For a Pizza Restaurant”



5.1 Visualization Demo: Surfacing Actionable Recommendations

Figure 8: Visualization Demo”



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

5.1.1 CRM System Integration Mockup

On the left of Figure 8, we show a conceptual UI for how this recommendation module could be embedded in the Heymate CRM by the Heymate engineering team. This allows restaurant managers to easily access insights from their dashboard, specifically under the “Recommendation” tab, alongside other operational menus.

5.1.2 Recommendation Output Visualization

On the right of Figure 8 is an Altair-based bar chart showcasing the top dishes sorted by computed popularity score. Key design features include:

- Top 3 highlights using Heymate brand colours (yellow, red, rose).
- Pop score bars are annotated directly with percentage labels.
- Hover tooltips for dish name and score for ease of exploration.
- Combined label (dish base + flavor) for intuitive comprehension.

This visualization not only validates that our scoring pipeline works but also provides a clear and professional way for stakeholders to compare and act on dish performance. It is also exportable as a standalone HTML component and ready for dashboard integration.

6 How to Use the Data Product

We designed this data product to support Heymate's internal team and clients in making data-informed menu decisions.

6.1 Intended Usage

Heymate can utilize this tool in two primary ways:

- Internal Use Mode:
  - Data Update & Validation: When a new data source becomes available, the internal team can upload and process it through the system to evaluate performance with updated inputs.
  - Testing & Iteration: By inputting internal client restaurant types, internal users can assess the quality of dish name cleaning and observe how the system generates standardized output. This enables iterative refinement and ensures reliability before onboarding new clients.
- Client-Facing Deployment: On the frontend, a client can simply select their restaurant type (e.g. "pizza restaurant"), and the system returns a list of top recommended dishes based on aggregated popularity metrics from similar restaurants. This empowers merchants to make data-driven menu decisions aligned with current market trends.

6.2 Strengths of the Data Product

1. Automated Cleaning & Standardization: GPT-powered semantic cleaning reduces manual effort in processing noisy menu data.
2. Scalability: Batch ingestion with Flask routing and SQL Server integration allows seamless future expansion as data volume grows.
3. Domain-Specific Recommendations: The system leverages aggregated restaurant-type-level patterns, delivering tailored suggestions relevant to each merchant.
4. Plug-and-Play Interface: Simple inputs (restaurant type) return actionable recommendations without requiring technical knowledge.

7 Data Science Methods

We applied many data science techniques in our capstone project and are highlighting 3 here:

- LLM Integration
- Distributed Deployment
- Materialized View

7.1 LLM Integration

We leveraged large language models (LLMs) to clean and standardize the menu data. The raw data was not available for direct use in our recommendation system due to inconsistencies in formatting, spelling variations, and the presence of multiple languages. By integrating LLMs, we were able to extract key information of dish bases and dish flavors, from item names, categories, and descriptions with high accuracy. Even in the case when the description is missing, the LLMs can still infer the information from the item name and category, as well as the context of the restaurant name.

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.
- Locally Deployed LLM: This could reduce long-term costs, but setup and maintenance would bring practical challenges within our capstone timeline.

7.2 Distributed Deployment

Our project involved processing a large dataset—over 3 million rows from the Uber Eats dataset. Cleaning this data sequentially would have taken approximately 5,000 hours, which was not feasible within our timeline. To solve this, we implemented a distributed deployment infrastructure to significantly speed up processing. We designed and deployed an HTTP-triggered function to process data in batches. Each instance is triggered via a web request, with the batch range passed as parameters. The system logs the task status at both the start and end of execution. To monitor tasks internally, we also built a dashboard in Looker Studio. Initially, we planned to deploy using Azure Functions on our partner's cloud infrastructure. However, due to security configuration challenges, we were unable to proceed with this plan. As a fallback, we deployed the system locally using the Flask framework and ran up to 20 worker instances concurrently, achieving a 20x speedup.

7.2.1 Limitation

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

7.2.2 Alternative Approaches Considered

We evaluated other cloud deployment solutions, such as EC2 from Amazon Web Services and Google Cloud Functions from the Google Cloud Platform. However, our partner uses Microsoft Azure, and we prioritized consistency within that ecosystem. In the future, our partner's engineering team plans to migrate our local deployment to Azure Functions, once security configurations are in place.

7.3 Materialized View

A Materialized View (also known as a Persistent Derived Table) is a database optimization technique that stores the result of a query as a physical table. This allows for much faster data retrieval by avoiding repeated computation over large datasets. Initially, it took around 6 minutes to generate a restaurant recommendation due to the volume of data. Such delays are unacceptable in production, especially since the recommendation module will eventually be integrated into the partner's CRM system. To optimize query performance, we implemented a materialized view to cache pre-computed results. This optimization reduced runtime from 6 minutes to just 3 seconds.

7.3.1 Limitations

- Additional Storage Cost: The materialized view aggregates data from the original 3-million-row table, incurring some storage cost. However, this is minimal relative to the base data and is not a major concern.
- Maintenance and Updates: When new data is ingested into the model input pipeline, the materialized view must be refreshed. To handle this, we established an automated workflow that updates the materialized view upon the successful completion of each data ingestion task.

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:

1. LLMs are transformer-based models and are not well-suited for handling structured logic or computations involving large-scale tabular data.
2. Interpretability is low: these systems often function as black boxes, making it difficult to understand or explain how the recommendations are generated.
3. 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, 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:** Our current popularity scoring system relies on static metrics, such as item frequency or appearance across menus, due to the lack of real transaction data. This makes it a proxy measure and may not fully reflect true customer preferences. The score could be further improved by incorporating additional quantitative features like price, review volume, or order frequency if available in the future.
- **Static Reference:** Our analysis is currently based on static snapshots of menu data. Without timestamped records, we are unable to capture how item popularity or menu composition evolves. This limits our ability to detect trends such as seasonal specials or emerging bestsellers.
- **Hard Filtering:** Restaurant type filtering is an exact match only (e.g., "pizza restaurant"), limiting support for broader or fuzzy queries like "comfort food" or "date night meals". A more flexible filtering mechanism could improve the versatility of the tool.
- **Evaluation Challenge:** Due to limited deployment and lack of real-time feedback from users, our validation process is based on illustrative case studies (e.g., the pizza restaurant example). While useful for demonstration, A/B testing or real user feedback would be necessary to confirm the real-world effectiveness of the tool.
- **Scalability Constraint:** Although our local pipeline performs well, we haven't deployed it to Azure yet due to resource and security constraints. As a result, the system is not yet scalable enough for real-time or production use cases.

- **Lack of Granular Personalization:** While our system standardizes restaurant types (e.g., "Chinese restaurant"), it does not yet support finer-grained classification at the sub-type level (e.g., "Szechuan," "Cantonese"). Additionally, the recommender ignores important factors such as local popularity, pricing, and user-specific preferences. These limitations reduce the system's ability to deliver tailored insights for diverse audiences.

9.4 Recommendations for Heymate

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

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

Our product lays a solid foundation for **scalable, transparent**, and **explainable menu 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.

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

Sreekala, Keshetti. 2020. "Popularity-Based Recommendation System." https://www.researchgate.net/publication/355773477.