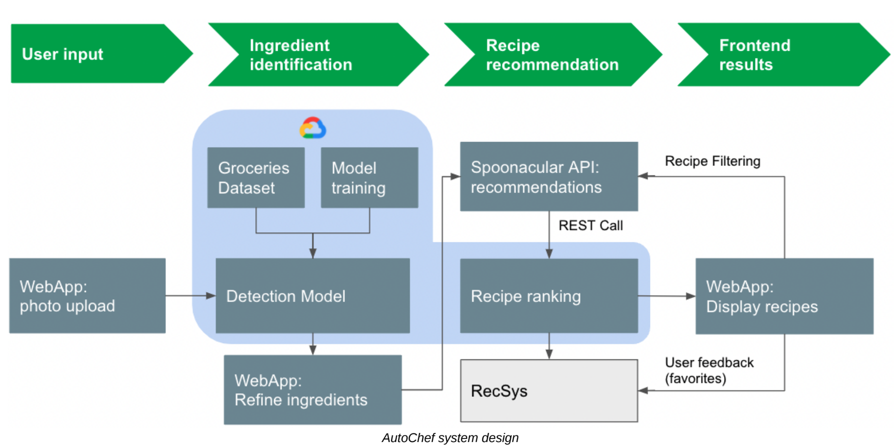
AutoChef: Computer Vision for Automated Ingredient-to-Recipe Matching

Problem Definition

When many of us go grocery shopping to our favorite convenience store, we have an idea of what we want to cook for the week and shop for the required ingredients accordingly. However, after cooking our intended meals for the week, more often than not, we have to deal with leftover ingredients. Often, these leftover ingredients sit in the fridge until they are ultimately thrown out, leading to food waste. These leftovers are tedious to identify one-at-a-time in the fridge, and sometimes, are completely unrelated: this week, Sharan - one of our team members - was left with kale, garbanzo beans, kimchi, lamb chops, and tuna. Moreover, this act of food waste reaches further than simply inconveniencing individuals - it is also a nationwide, environmental issue: Feeding America estimates that nearly 40% of all food in the U.S. is thrown out [1].

To help users solve this bothersome, yet relevant problem in their cooking lives, we developed AutoChef: a web application that automatically identifies leftover ingredients and recommends recipes that maximize usage of leftover ingredients. Unlike pre-existing recipe APIs such as SuperCook, which require users to manually figure out and type in leftover ingredients, AutoChef quickly identifies multiple ingredients from a single photo, allows users to adjust the identified list of ingredients, and recommends recipes based on cuisine, dietary restrictions, type of dish, and intolerances. Furthermore, AutoChef acts as a one-stop-shop for all cooking-related needs by also allowing users to favorite recipes and providing detailed recipe instructions.

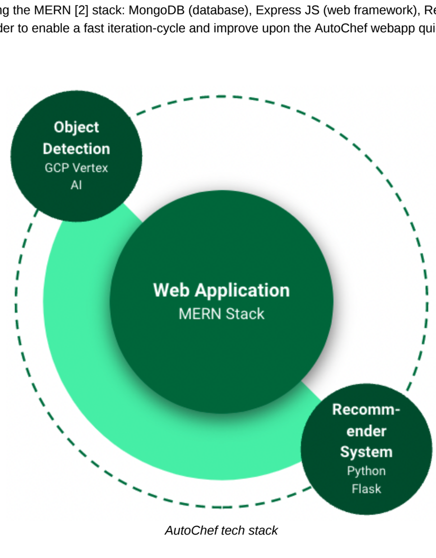
System Design



AutoChef system design

Web Application

The AutoChef webapp was created using the MERN [2] stack: MongoDB (database), Express JS (web framework), React JS (client-side framework), and Node JS (server-side framework). In order to enable a fast iteration-cycle and improve upon the AutoChef webapp quickly, we made use of Material UI for the AutoChef frontend.



AutoChef tech stack

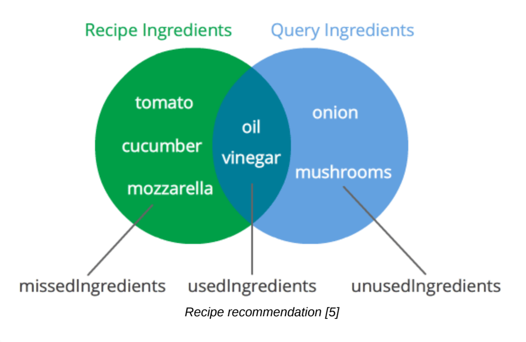
Ingredient Identification

The ingredient identification pipeline takes as input a single image uploaded by the user via the AutoChef webapp interface. Next, the object detection model, outputs the bounding boxes/associated classes of all the detected ingredients. The ingredient detection model was trained on the MVTEC Supermarket dataset [3] and deployed using AutoML on GCP Vertex AI [4]. Vertex AI allowed us to quickly train models without writing redundant training code for object detection. Furthermore, Vertex AI quickly evaluates false positives/negatives for each ingredient label, helping us understand which ingredients were harder to detect. Finally, we chose to deploy it on Vertex AI for ease-of-access from multiple instances of the AutoChef webapp. The webapp communicates with the model to provide an image input and obtain a list of detected ingredients as output using REST API calls.

The object detection model achieves a Mean Average Precision (MAP) of 0.871, which is sufficient for practical use with everyday ingredients as the model mostly fails in cases where the ingredients to be detected are rare. In order to deal with undetected/miclassified ingredients, we parse the list of detected ingredients as checkboxes that are displayed on the webapp interface. Misclassified ingredients can then be unselected using these checkboxes by the user. Furthermore, the interface contains a textbox for users to input additional ingredients to be incorporated into the recipe. Once the final list of ingredients have been determined by the user, this list is then passed on to the Recipe Recommendation System.

Recipe Recommendation

The Recipe Recommendation component can be divided into two components: Ingredient-to-Recipe Matching and Recipe Ranking.



Recipe recommendation [5]

Ingredient-to-Recipe Matching

Ingredient-to-recipe matching is driven by the Spoonacular API [5], which is a third-party food and recipe API. Among Spoonacular's wide-ranging functionality, we primarily used the `complexSearch` API endpoint: `GET https://api.spoonacular.com/recipes/complexSearch`. The inputs are:

1. `query` (string): comma separated list of ingredients (Ex: `lettuce, tomato, ...`)
2. `cuisine` (string): comma separated list of cuisines (Ex: `american, greek, ...`)
3. `diet` (string): dietary restrictions (Ex: `vegetarian`)
4. `intolerances` (string): comma separated list of intolerances (Ex: `gluten, dairy, ...`)
5. `type` (string): meal type (Ex: `dessert`)

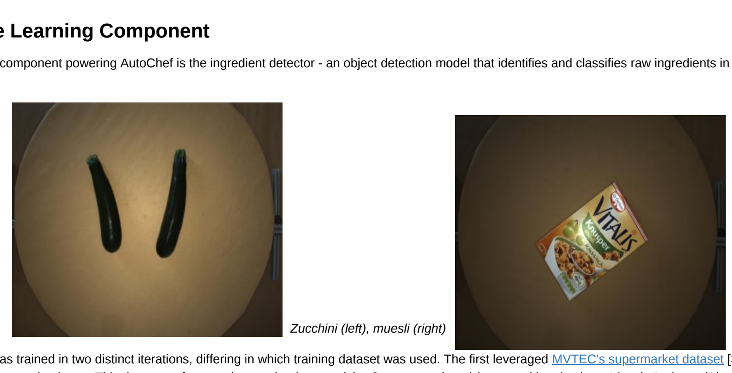
The `query` string is obtained from the Ingredient Detection component whereas the remaining recipe filters are obtained from the user through the webapp interface.

Recipe Ranking

The Recipe Ranking component is deployed on Python flask, which accepts user inputs from the webapp that are parsed to form the API `GET` request parameters. The recipes were sorted by: 1) ascending # missed ingredients, 2) descending # used ingredients, 3) descending Spoonacular recipe score, and 4) descending # likes. The list of recipes are then JSONified and finally sent back to the webapp to display results.

Machine Learning Component

The core ML component powering AutoChef is the ingredient detector - an object detection model that identifies and classifies raw ingredients in an image.



Zucchini (left), muesli (right)

This model was trained in two distinct iterations, differing in which training dataset was used. The first leveraged MVTEC's supermarket dataset [3] -- a public dataset of supermarket items. This dataset unfortunately contained many niche German products/classes, with only about 10 truly "universal" ingredients (eg. zucchini). For the sake of building a market-ready application, this dataset lacked diversity but was a good starting point. Two sample images from this dataset are shown:

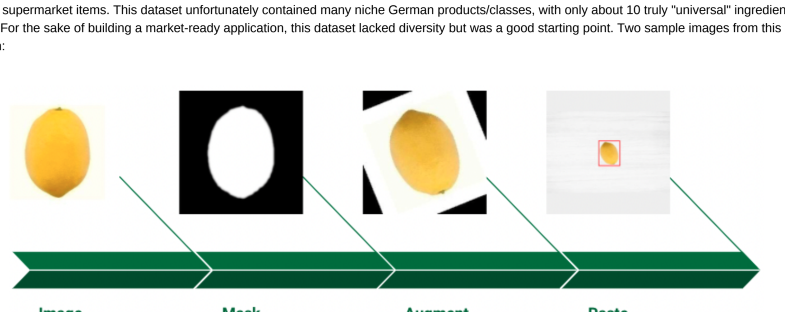
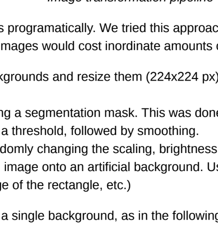


Image transformation pipeline

The second dataset was curated by synthesizing images programatically. We tried this approach because searching for a dataset of raw ingredients was fruitless (pun intended), while creating a dataset of real images would cost inordinate amounts of resources. As such, we took the following steps:

1. Crawl the web for ingredient images with white backgrounds and resize them (224x224 px). In total, we obtained ~100 images across 20 classes, with 4-7 items per class.
2. "Remove" the background of each picture by creating a segmentation mask. This was done by adapting the code from this Stack Overflow post [6]. In short, we convert the image to grayscale and apply a threshold, followed by smoothing.
3. Artificially transform the image and the mask by randomly changing the scaling, brightness, saturation, and rotation.
4. Using the transformed mask, paste the transformed image onto an artificial background. Use the coordinates of the transformed mask to compute the bounding box (i.e. leftmost white pixel is the left edge of the rectangle, etc.)

When creating this dataset, we pasted many items onto a single background, as in the following example:



Sample training example from our synthetic dataset

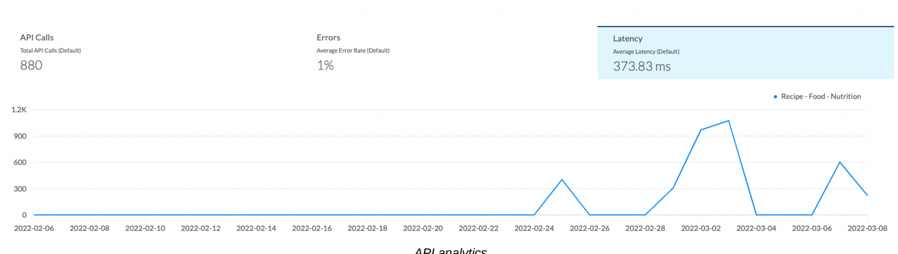
The remaining model creation process is identical for both. To train each model, we saved the images in GCP and curated a dataset with an annotation file to identify bounding boxes. We then trained the model with AutoML [4], which completes training end-to-end -- abstracting away the architecture search and hyperparameter tuning.

System Evaluation

Web Application

The webapp was primarily tested on handling large API call loads when many users use the application concurrently. 100 instances of the webapp were spun-up and repeatedly made five Spoonacular API calls to evaluate the impact on ingredient-to-recipe matching latency. During this peak load, a total of 500 API calls were made (5 from each of the 100 instances) and the latency jumped to ~1,000 ms from the typical single API call latency of ~250 ms. Therefore, there was a 4x latency increase corresponding to a 100x user increase. While this is concerning if AutoChef is to grow to support a multitude of concurrent users, this issue can be dealt with using API load balancers. Unfortunately, we chose not to add to the webapp at this time due to limitations in expenditure.

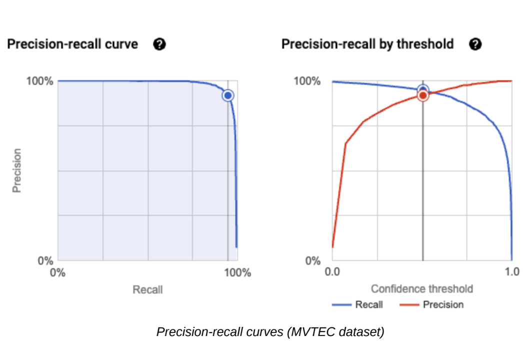
Additionally, the AutoChef webapp was tested by a variety of users for bugs. Several bugs were encountered during beta-testing, all of which were fixed.



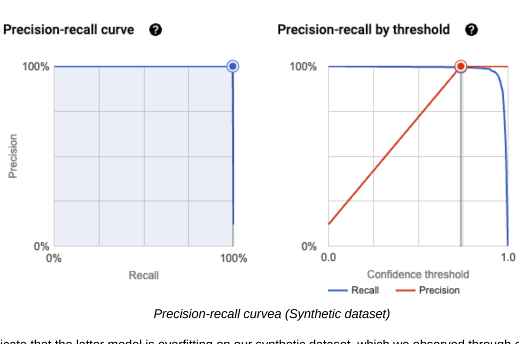
API analytics

Ingredient Identification Evaluation

The ingredient identification models were evaluated by comparing Mean Average Precision (MAP) on GCP precision/recall curves. The MAP for the MVTEC dataset was 0.871, and 0.905 for the synthetic dataset.



Precision-recall curves (MVTEC dataset)



Precision-recall curvea (Synthetic dataset)

The precision-recall curves indicate that the latter model is overfitting on our synthetic dataset, which we observed through empirical evaluation. This is not particularly surprising -- our synthetic dataset contains only 1000 images of about 100 unique ingredients. Even after augmentation, this is insufficient and we will need a massive, diverse dataset of images with assorted foods.

Recipe Ranking evaluation

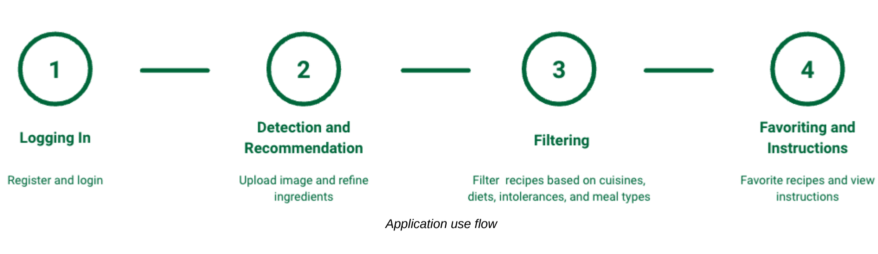
Unfortunately, given the limited time/resources, we could not comprehensively evaluate our recipe recommendation system. However, we tested a variety of ingredients and filters to find edge-cases (i.e no relevant recipes given the recipe filters). We also experimented with a variety of metrics and their priority orders to sort recipes on, in order to find relevant recipes.

That being said, several limitations remain with the recipe recommendations. Firstly, we could implement more of a personalized recommendation system that also uses data on users' past favorited recipes to recommend similar recipes. We could further improve this approach through the use of collaborative filtering [7] on favorited recipes across users. Secondly, we could perform a more objective verification of our recommendations through the use of Amazon Mechanical Turk to verify that the recipes recommended are actually relevant. Finally, we could calculate more objective metrics (i.e. percentage of users satisfied with top 10 recommendations) for improved recipe recommendation.

Finally, it is worth noting that our system will naturally improve as more users use the Spoonacular API [5]. We noticed that recipe relevance and popularity were heavily correlated to Spoonacular-specific metrics (i.e. high Spoonacular score, high # of aggregate likes, etc.). Making more `GET` requests for specific recipes increases their Spoonacular score, thus boosting their relevance, which allows these recipes to be ranked at the top. Therefore, as the Spoonacular API and the applications that support the API gain more users, the recipe ranking system will improve.

Application Demonstration

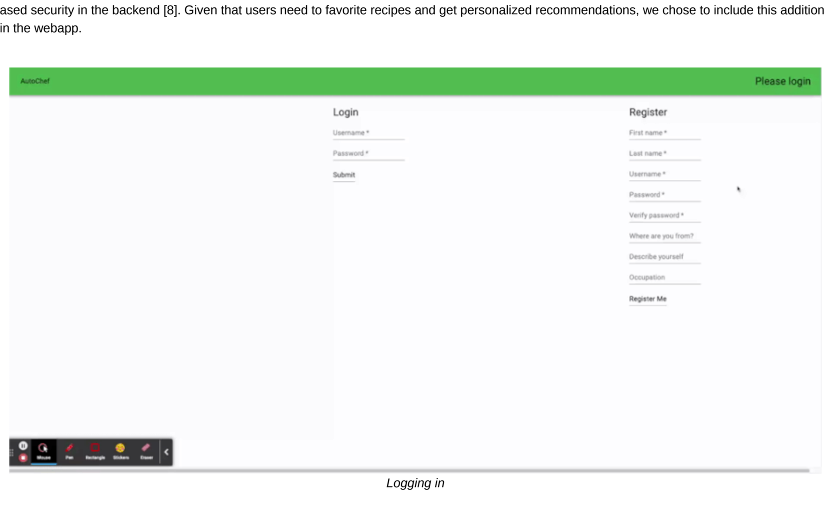
The webapp consists of four steps for accessing all of its functionality:



Application use flow

1. Logging In

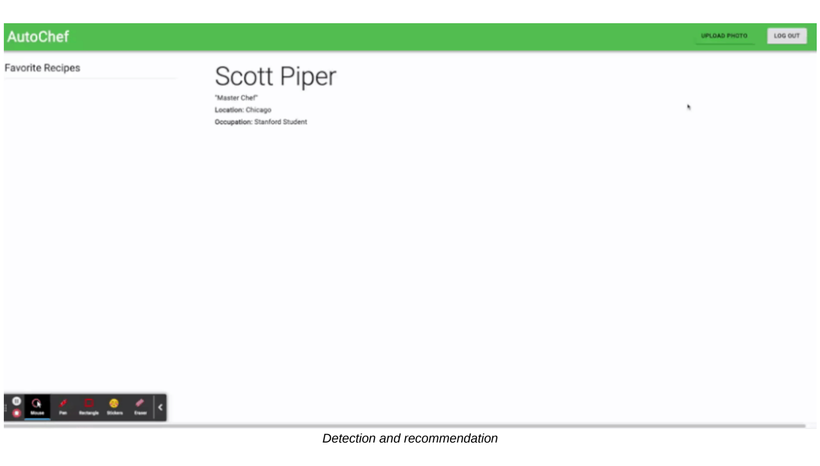
A user can register using the textboxes provided on the frontend and then proceed to login using the same. The passwords are salted and hashed for increased security in the backend [8]. Given that users need to favorite recipes and get personalized recommendations, we chose to include this additional step in the webapp.



Logging in

2. Detection and Recommendation

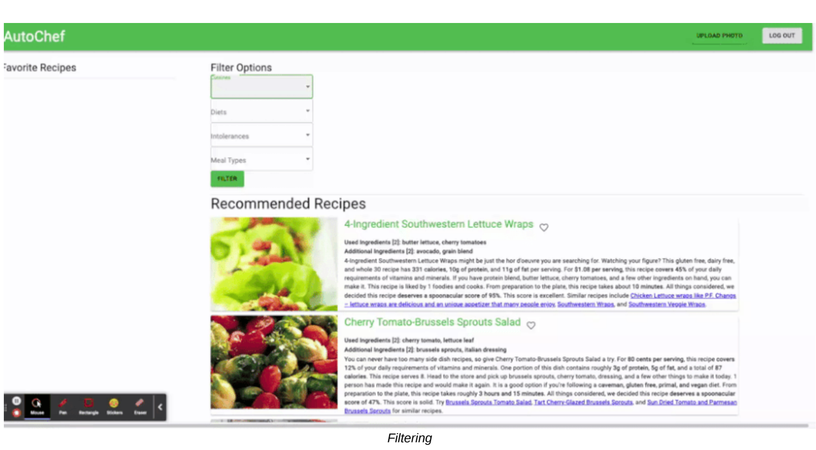
An image containing all the ingredients is uploaded using the frontend, which is then passed as input to the object detection model deployed on GCP Vertex AI [4] that returns a list of detected ingredients via the backend. A textbox is included to manually add ingredients, which are then parsed as checkboxes on the frontend to deal with undetected/misclassified ingredients. A checkbox for incorporating common pantry items (Ex: salt, milk, etc.) is also included for ease-of-use. Clicking on the recommend recipes button makes the `GET` request to the Spoonacular API via the backend, which then returns a list of suggested recipes to be displayed on the frontend.



Detection and recommendation

3. Filtering

Recipes can be filtered using the four filter dropdown boxes on the frontend: cuisines, diet, intolerances, and meal type. The cuisines and intolerances dropdown boxes comprise of checkboxes since multiple options can be selected. Clicking on the filter button then filters out the displayed recipes. If no filter options are selected, then no filter options are applied.

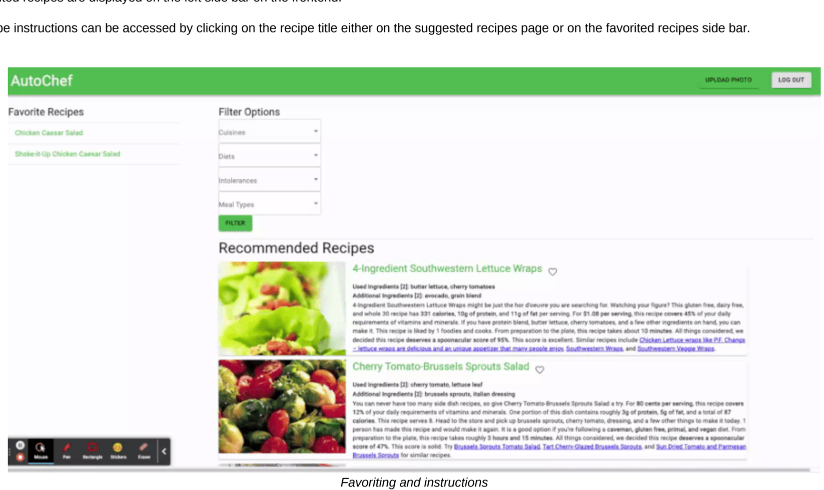


Filtering

4. Favoriting and Instructions

Recipes can be favorited and saved in the database (for future use to save making additional API calls) using the heart icon beside the recipe titles. A user's favorited recipes are displayed on the left side bar on the frontend.

Recipe instructions can be accessed by clicking on the recipe title either on the suggested recipes page or on the favorited recipes side bar.



Favoriting and instructions

Reflection

What worked

Web Application

Given the team's proficiency with developing webapps, designing AutoChef as a webapp was the natural choice. Furthermore, the webapp allowed for easy integration with the ingredient detection and recipe recommendation systems, both of which were deployed on different servers.

MVTEC Dataset

The MVTEC dataset [3] was perfect for AutoChef as it contained a large number of examples and classes with images captured under diverse lighting conditions. The model generalized well to real-world examples during our testing. The only limitation was the number of useful classes as it contained a lot of ingredients only available in German supermarkets.

Spoonacular API

The Spoonacular API [5] was the core driver in the ingredient-to-recipe matching step. A plethora of suitable APIs were considered: MyCookBook, Tasty, Edamam, Zestful, Yummly, and TheMealDB. We decided to choose Spoonacular as it provided the best functionality-to-cost trade-off. Furthermore, it contained the largest selection of ingredients (2,600+), recipes (5,000+), products (90,000+), and menu items (115,000+) among the APIs considered.

What did not work

Synthetic Object Detection Dataset

The synthetic dataset was curated to fine-tune our object detection model to improve performance on Western ingredients. Despite being curated programatically, this process is not scalable:

1. Parsing through the Google search results to ensure the images are high-quality and realistic required a human-in-the-loop.
2. Pictures with white backgrounds (which are rare) are required to segment and paste the images on realistic backgrounds as Google search images do not contain bounding boxes.

Given these constraints, the synthetic dataset curated was small, resulting in the model to quickly overfit on the dataset.

ML-based Recommendation System

The recommendation system can be drastically improved and personalized to leverage app usage using ML techniques. Similarity metrics such as cosine similarity [9] or siamese networks [10] can be used on recipe embeddings obtained using language models on recipe instructions and other features for comparing a user's favorited recipes to recommended recipes. Furthermore, as AutoChef scales, collaborative filtering [7] approaches and latent factor models [11] can be used across our users for recommending more personalized recipes.

Next steps

Curate a Better Training Dataset

The most labor and cost intensive, yet crucial next-step in improve ingredient detection would be to curate a better dataset, designed to serve a more Western population.

Integration

Given the time constraints, the team decided to divide and parallelize development without much thought of future integration. As a result, we ended up with three systems, each running on different servers: webapp (Node JS), ingredient detection (GCP Vertex AI), and recipe recommendation (Python flask). Integrating the ingredient detection and recipe recommendation into the webapp to run under the same server would allow for fast latency.

Mobile Application

AutoChef's use-cases are primarily intended for mobile apps since mobile phones can be used to easily take pictures of ingredients. While hosting the webapp on a public server allows for access via mobile phones, the experience is not as smooth as it is on a computer. Designing a more mobile-friendly version or perhaps even an iOS or Android application to run natively on mobile phones would be the next step in scaling AutoChef to its next million users.

Broader Impacts

AutoChef was built to be a one-stop-shop for users to get recipe recommendations and instructions seamlessly using a simple photo of their ingredients. While we expect minimal intentional and/or unintentional harmful usage of AutoChef, potentially harmful consequences could exist.

AutoChef's recipe recommendation is biased to suggest recipes that are largely Western cuisine focused and as such, it may have a difficult time detecting ethnic ingredients (Ex: kombu, asafoetida, etc.). In order to mitigate this, we ensured to include diverse cuisine filters that are non-Western that the user could select to suggest recipes that are relevant to them.

Additionally, dietary restrictions are an important aspect for many users due to food allergies or conscious food choices. A misclassification from the object detection model could thus lead to severe consequences for AutoChef users. While it is highly unlikely that a user will consume a misclassified ingredient simply due to a recipe recommendation, we parse the ingredients as checkboxes that allow users to manually select ingredients. This enables users to selectively choose leftover ingredients as they wish, instead of solely relying on the model and its full list of detected ingredients in recommending recipes.

References

[1] "How we fight food waste in the US," Feeding America. [Online]. Available: https://www.feedingamerica.org/our-work/our-approach/reduce-food-waste#:~:text=Shockingly%2C%20nearly%2040%25%20of%20all,billion%20pounds%20of%20food%20waste. [Accessed: 09-Mar-2022].

[2] "How to use MERN stack: A complete guide," MongoDB. [Online]. Available: https://www.mongodb.com/languages/mern-stack-tutorial. [Accessed: 09-Mar-2022].

[3] Follmann, P., Böttger, T., Härtinger, P., König, R. and Ulrich, M., 2018. MVTec D2S: Densely Segmented Supermarket Dataset. [online] arXiv.org. Available at: https://arxiv.org/abs/1804.08292 [Accessed 10 March 2022].

[4] Vertex AI. [Online]. Available: https://cloud.google.com/vertex-ai. [Accessed: 09-Mar-2022].

[5] "Spoonacular API Documentation," spoonacular recipe and food API. [Online]. Available: https://spoonacular.com/food-api/docs. [Accessed: 09-Mar-2022].

[6] Stack Overflow (2020). how to remove background of images in python. Retrieved 10 March 2022, from https://stackoverflow.com/questions/63001988/how-to-remove-background-of-images-in-python

[7] R. Zhang, Q. -d. Liu, Chun-Gui, J. -X. Wei and Huiyi-Ma, "Collaborative Filtering for Recommender Systems," 2014 Second International Conference on Advanced Cloud and Big Data, 2014, pp. 301-308, doi: 10.1109/CBD.2014.47.

[8] Gauravaram, P. (2012, November). Security Analysis of salt|| password Hashes. In 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT) (pp. 25-30). IEEE.

[9] H. Khatter, N. Goel, N. Gupta and M. Gulati, "Movie Recommendation System using Cosine Similarity with Sentiment Analysis," 2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA), 2021, pp. 597-603, doi: 10.1109/ICIRCA51532.2021.9544794.

[10] Angelovska, M., Sheikholeslami, S., Dunn, B., & Payberah, A. H. (2021, April). Siamese Neural Networks for Detecting Complementary Products. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop (pp. 65-70).

[11] J. Fang, X. Zhang, Y. Hu, Y. Xu, M. Yang, en J. Liu, "Probabilistic Latent Factor Model for Collaborative Filtering with Bayesian Inference", CoRR, vol abs/2012.03433, 2020.