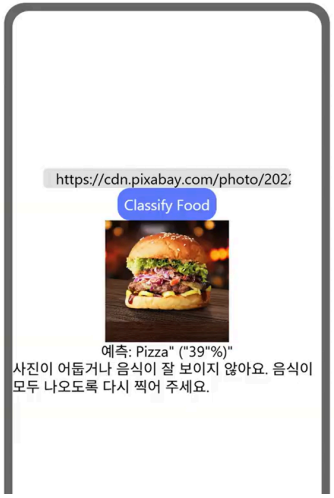
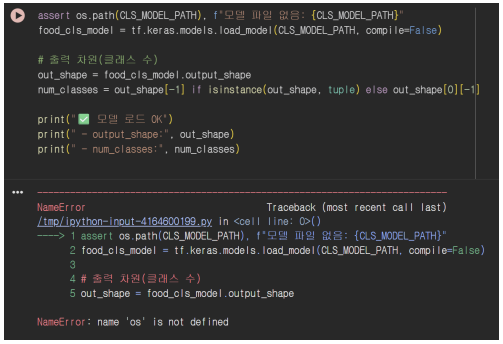
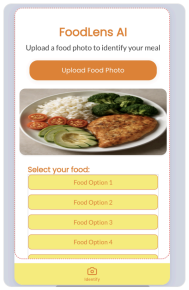
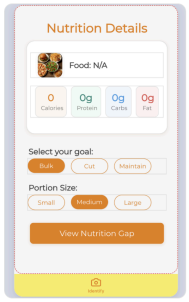
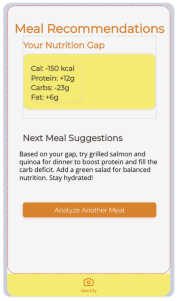
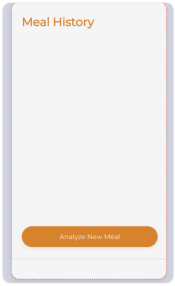
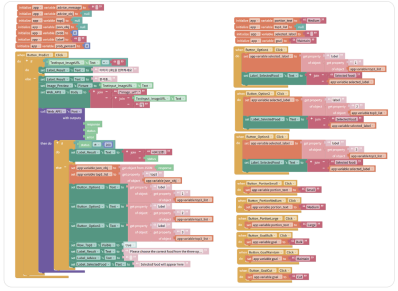
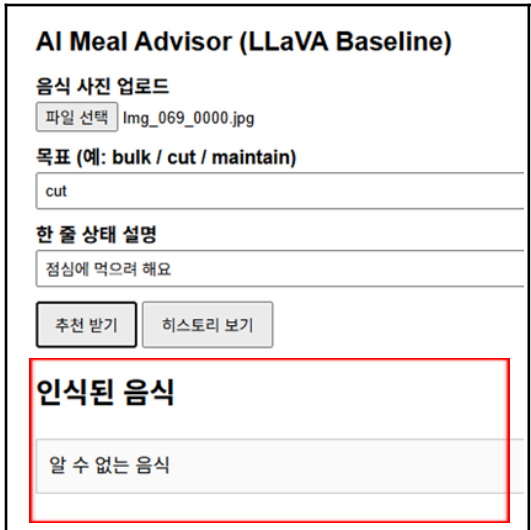
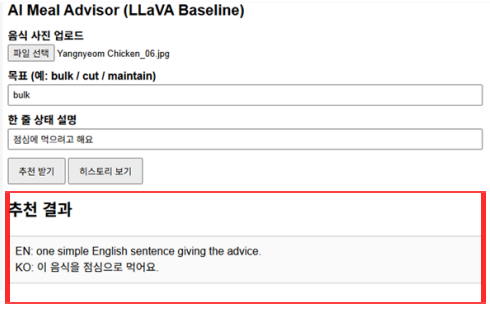
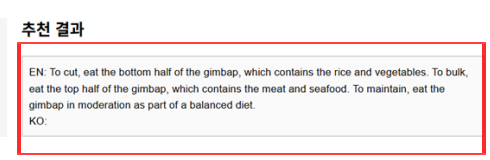
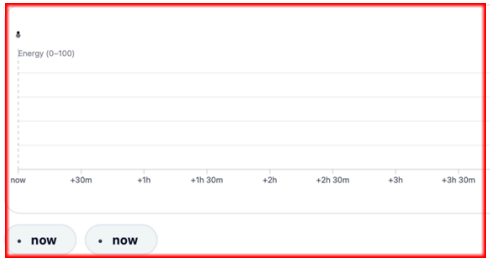
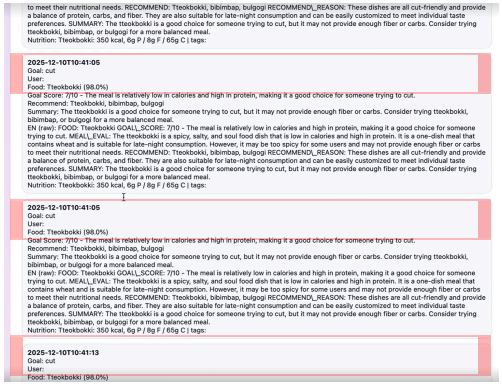
Error Board

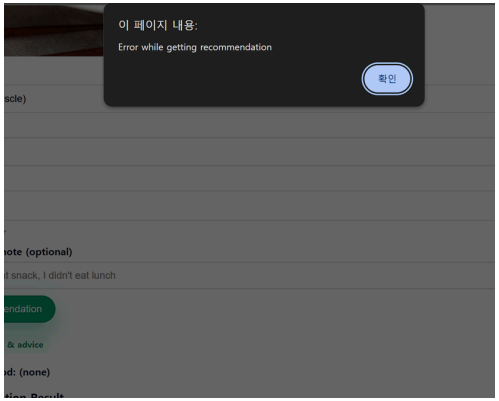
Input Image(The Failure)	Expected Output vs Actual Output	Hypothesis (Why did it fail)	Fix, Action (What did you do, added more data, changed background)
	Expected Output : It will recognize the photo as a hamburger. Actual Output : It recognized the photo as pizza (39%).	The model sees both hamburgers and pizza as foods with a round shape and bright toppings like cheese, bread, and sauce, so it gets confused between them. Especially in photos taken from above, the hamburger's bun and patty structure is not clear, so it may look similar to the flat shape of pizza. This is not just a problem of not having enough data, but a generalization problem where the model did not learn the structural differences between the classes well enough.	To solve this problem, instead of just adding more images, we rebuilt the dataset focusing on images that clearly show the structural differences between hamburgers and pizza. Specifically, we added side-angle images where the hamburger's layered structure (bun–patty–filling) is clearly visible, separated the data so it is distinct from top-view images that emphasize pizza's flat shape, and diversified the backgrounds so that plates, wrappers, and background colors do not cause confusion. Through this, we guided the model to classify based on the three-dimensional structural features of the food, not just color or shape, and improved it to reduce misclassification between hamburgers and pizza.

	Expected Output: Various real food images are correctly classified into each food type. Actual Output: Food images that were not learned are wrongly classified into the most similar existing food class or recognized as unknown.	The initial model was trained on a dataset limited to 21 types of food, so it could not generalize well to the various forms and types of food that appear in real environments. Because of this, when the model saw food images that were not included in the training data, it did not recognize them as new foods, but instead forced them to match the most similar existing class or treated them as unknown class. We decided that the main causes were the lack of food variety and the limited class coverage.	To solve this problem, we greatly expanded the food image dataset. Moving beyond the original 21 types of food, to better reflect real user environments, we expanded the food types to a total of 300 food categories and collected additional images of various shooting conditions (lighting, angles, backgrounds) for each food. Through this, the model did not rely too much on specific foods, learned a wider range of food images, and improved its generalization performance.
	Expected Output: Since we completed file validation (clean-up) for all train/validation images in advance, we expected TensorFlow to load all images normally during the training process and proceed without errors. Actual Output: Image loading errors occurred repeatedly during training, and in some runs, training stopped or prediction results appeared abnormal.	Even though file format and name validation at the folder level was completed, the problem continued, so we determined that the cause was in specific image files that looked normal on the outside but were damaged internally. In other words, we made a hypothesis that the file extension and size were normal, but during the process of TensorFlow decoding or resizing (240×240), specific images were read in a broken state and caused errors. We also considered the possibility that these problematic images caused occasional loading failures during the process of reading images from the Google Drive environment.	To solve the problem, we went beyond simple folder validation and applied a method to directly remove the images that caused errors during the training process. Specifically, we checked the timing and logs when errors occurred during training to track images that could be problematic, removed those images from the dataset, and ran the training again. As a result, the image loading errors that occurred repeatedly before disappeared, and we confirmed that model training completed stably. Through this, we confirmed that the cause of the problem was not in the code or overall data structure, but in some damaged image files.

    	Expected Output: Initially, we aimed to implement the service in mobile app form to stably provide image input and result output in one integrated app environment. Actual Output: Various errors that occurred in the app environment and the complexity of connecting the multimodal model caused significant difficulties in development and debugging. As a result, implementation stability and development speed decreased much more than expected, and it became difficult to fully implement the service functions.	We determined that the cause of the problem was that the app environment itself was too complex to handle multimodal large models. Especially to directly connect an MLLM like LLaVA to an app, we had to manage multiple elements at the same time, such as model size, inference resources, server-client communication, and debugging environment. Because of this, when errors occurred, it was difficult to quickly track the cause. In other words, we determined that the key cause was not that app implementation was technically impossible, but that it was an excessive structure compared to the project scope and available resources.	To solve this problem, we changed the service implementation method from a mobile app to a web-based structure. The web environment had advantages of easy debugging, intuitive connection with Colab, and quick verification of the model inference process. Also, we chose LLaVA as the multimodal model. LLaVA is the most standard open-source MLLM, has excellent image description performance, and has many implementation cases and references, making stable implementation possible. In addition, by using an A100 GPU that provides compatibility with the LLaVA model and sufficient computing resources, we could run the large-scale multimodal model more stably and minimize errors caused by memory shortage or computation bottlenecks. Through this, we secured the stability of the LLaVA-based image analysis and text generation process and improved the performance and reliability of the entire system.

	Expected Output: Since image training and testing completed normally in the Colab environment, we expected that when we loaded the model into the Colab environment for the website, the food images uploaded from the web would be recognized the same way and the recognized food name would be displayed normally on the web screen. Actual Output:Even though image upload and model call worked normally, a problem occurred where the food was not recognized or the result value was displayed empty.	We determined that this problem likely occurred due to pipeline inconsistency in the web-Colab connection process rather than the model performance itself. We hypothesized as the main causes that the image sent from the web was preprocessed differently from the input format used during training in Colab, or that the image received from the web was passed to the model inference function, but there was a problem with the parsing or output logic in the process of returning the result value to the web frontend. In other words, we determined that the model existed but the input-output connection was not fully connected.	We debugged the web-Colab inference pipeline step by step. First, we verified image uploads through logs and aligned preprocessing (resize, normalization) with training settings. Next, we reorganized the label mapping logic to accurately connect model outputs with the web's food name list. Finally, we specified the output format to prevent empty returns and added debugging messages. Additionally, we implemented a manual override feature, allowing users to directly input food names as an alternative option. This resolved the issue, ensuring recognized food results display correctly on the website.
	Expected Output: Based on the food image and the goal selected by the user, one specific sentence of dietary advice in English would be generated and a Korean explanation would be provided together with it. Actual Output: The English advice was not generated and only a guide message was displayed, and the Korean also only output a fixed sentence unrelated to the food or goal.	This problem appears to have occurred because the model or API call that generates the recommendation sentence was not completed normally. As a result, the default template text was displayed on the screen instead of the actual inference result.	We added validation logic to check whether the recommendation result was generated normally, and modified it to not display the default text when there is no result. We also improved the UI to clearly display a demo guide message when the server or model is in an inactive state.
	Expected Output: The expected output was that only the dietary advice corresponding to the one goal selected by the user (e.g., cut, bulk, maintain) would be displayed.	This problem appears to have occurred because the step that generates the recommendation sentence did not branch based on the user's goal value as a condition, but generated explanations for all goals as one text. In other words, the goal	We modified the logic to generate sentences based only on the goal value selected by the user when generating recommendation results. Also, on the frontend, we improved it to show only advice for one goal by filtering the results one more time so

	Actual Output: A sentence that included advice for bulk and maintain together was displayed at once.	filtering logic was not properly applied in the prompt or post-processing step.	that sentences that do not match the selected goal are not displayed.
Goal Score: 6.0/10 · Needs improvement **Your Meal Evaluator** This meal looks like 350 kcal. For cut, portion control + protein matters most. Context noted: - **Recommended Dishes** Chicken Breast Salad,Kimchi Stew,Ramen **Why these dishes?** **Summary** Main tip: keep protein steady and adjust calories by goal. Meal typ **Nutrition** Tteokbokki: 350 kcal, 6g P / 8g F / 65g C | tags: **Energy Curve Preview (next 4 hours)** This meal looks like 350 kcal. Context noted: -  3 Manual override	Expected Output: Since we wrote code to calculate and visualize the energy curve, we expected that the energy change by time or step would be generated normally as a graph according to the input data. Actual Output: A problem occurred where the graph was not displayed on the screen or only an empty graph was shown.	We determined that the cause of the problem was not the graph generation process itself, but that the calculated energy data was not properly passed to the visualization step. We hypothesized as the main causes the possibility that the energy value calculation result existed only as a local variable and could not be accessed by the visualization function, or the possibility that the rendering call to display the graph in the web/Colab environment was missing. In other words, we determined that the energy calculation logic and the graph output logic were not fully connected.	We checked the energy curve generation process step by step. First, we confirmed through intermediate output whether the energy values were actually accumulating normally in the list, and modified the variable scope to explicitly pass the data to the graph function. Also, we supplemented the code to ensure that the show or web rendering related function was not missing after graph generation. Through this, we organized the pipeline so that the calculated energy data properly continued to the visualization step, and as a result, we confirmed that the energy curve graph was generated normally.
	Expected Output: We predicted that in the history function of the web service, the uploaded food images would be recorded according to the time slot based on the time zone set directly by the user. Actual Output: Regardless of the time set by the user, the images were saved in the history based on the actual time they were uploaded to the server. This caused problems where the user's meal records were sorted differently than intended, or food was displayed in the wrong time zone.	We determined that the cause of the problem was that the user input time and the system automatically generated time (timestamp) were not clearly distinguished in the process of handling time information. When saving history items on the server or database, the upload time (timestamp) set as the default value was used first, and the time value selected by the user was not properly reflected in the storage logic. We also considered the possibility that the time information sent from the frontend was missing during the	To solve this problem, we improved the structure by modifying the history storage logic to save the time set by the user as a separate explicit field. On the frontend, we modified it to send the time selected by the user to the server in a clear format, and on the backend, we changed the logic to process that value separately from the upload time and use it as the basis for history sorting and display. Also, we separated the automatically generated upload time so that it only remains for logging and internal management purposes and does not

		parsing or mapping process on the backend.	affect the user screen. Through this, we were able to improve the history function to work according to the user's intention.
	Expected Output: Classify the food image into the matching class and provide a recommendation accordingly. Actual Output: An error occurred and no recommendation was provided.	We assume that the cell execution order in the Colab notebook was not correct, so the class labels or data paths that the model should reference were not loaded properly. Although we cannot rule out the possibility of an error in the code itself, after checking the code structure, we thought it was more likely a problem with the execution order rather than a logic problem. Also, when compared to the version that worked normally the day before, the only changed element was the cell execution order, so we determined that this problem was likely the main cause.	We rearranged the execution order of cells that set the data path and class order in the Colab notebook. Also, by comparing the previous version that worked well with the current version, we modified it so that the link settings and array order were the same. As a result, we confirmed that the food images were matched to the properly trained classes and provided recommendations accordingly.