Aldone

Fausto Zamparelli, Daniel Falbo

May 30, 2024

Introduction

Aldone as in ”AI-Done” is a tool for personal reminders and grocery list management. This is a proof of concept of a product that could save you time every day by allowing you to smoothly interact with it on the fly.

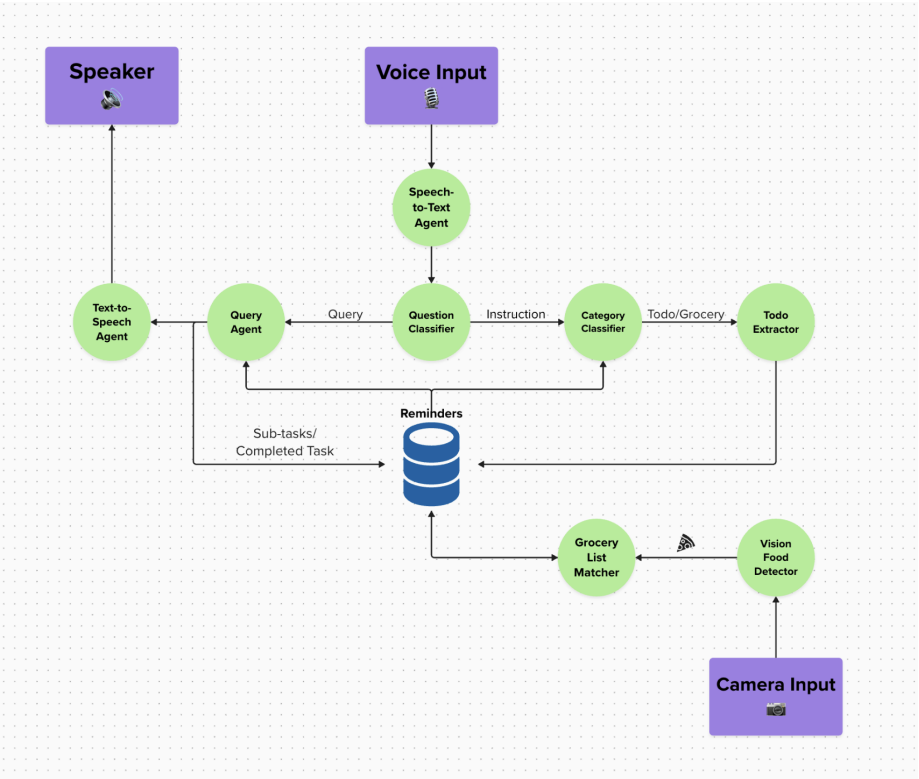
Aldone is a react web app that interacts with a node.js and a python server doing most of the magic. After a log-in you will be able to speak to it to add tasks to your grocery list, to-do list, or make questions about reminders you saved earlier. Furthermore, you could ask to split a task in sub-tasks or approximate how long a task will take. Aldone is able to detect fully by himself weather you are asking him to add something to your grocery list or to-do list.

The coolest thing is that once you will come home from the supermarket with tired legs and arms from carrying bags, you won't have to manually remove everything from this digital grocery list but you can use your phone camera to detect what foods you actually bought and then Aldone will remove them for you. Other use cases for this include checking what's already in your kitchen or marking products as bought after delegating your groceries to someone else.

Sources of Inspiration

1. The prof's demos and examples from the Google Classroom.
2. Cosmos: the "operating system" used by the Humane Pin, which is a new AI-first device designed to be used only with your voice and vision.
3. aiXplain: which is a fresh startup allowing to create complex AI pipelines by connecting independent AI agents with each other through a graphical drag and drop interface.

Method



Visual representation of the agents that make up Aldone

As we can see Aldone was created by splitting its behind-the-scenes functionalities into multiple specialized AI agents that cooperate with each others. In order to fully understand how Aldone works under the hood it is best to understand the functionality of each of these agents alone. Let's work our way thorough the graph from top to bottom:

Speech-to-Text Agent

For the Speech-to-Text task, being a web app, Aldone uses the browser's built-in Web Speech API service for convenience. The implementation can be found by searching for "webkitSpeechRecognition" in the 'src/app/[username]/page.tsx' file.

Question Classifier

The task of the Question Classifier is to determine whether the user's input is a question/query or a statement/instruction. Anything that involves looking for information in the web or from the existing data in the database, is a query. Anything that provides new data within the input, is not a query.

Input	Label
"What's the capital of Italy?"	Query
"Add milk to my grocery list"	Not Query
"Split task X into subtasks"	Query

For example, the input "What's the capital of Italy?" is a query, while "Add milk to my grocery list" is an instruction. The input "Split task X into subtasks" is also a query because the data is not being provided by the input.

The Question Classifier is implemented using a neural network trained on a dataset of 500 examples. The dataset was manually generated and enhanced using generative AI data synthesizing tools. The question classifier is a pytorch neural network and its code can be found in the subdirectory "src/py/question_classifier/".

In the subdirectory,

- `nndata.json`: is the dataset used to train the model
- `nntraining.py`: is the training script that generates the model. 80% of the data is used for training and 20% for testing. Of the training data, 25% is used for validation.
- `nnlabeler.py`: uses the trained model to label new input data.

```
❯ py nntraining.py
Epoch [1/10], Training Loss: 0.6744, Validation Loss: 0.6710
Epoch [2/10], Training Loss: 0.6527, Validation Loss: 0.6442
Epoch [3/10], Training Loss: 0.5999, Validation Loss: 0.6062
Epoch [4/10], Training Loss: 0.5131, Validation Loss: 0.5528
Epoch [5/10], Training Loss: 0.4322, Validation Loss: 0.4894
Epoch [6/10], Training Loss: 0.3818, Validation Loss: 0.4219
Epoch [7/10], Training Loss: 0.3134, Validation Loss: 0.3563
Epoch [8/10], Training Loss: 0.2414, Validation Loss: 0.2980
Epoch [9/10], Training Loss: 0.2089, Validation Loss: 0.2499
Epoch [10/10], Training Loss: 0.0988, Validation Loss: 0.2112
Accuracy on test set: 98.00%
```

The model achieved 98% accuracy on the test set.

Category Classifier

The task of the category classifier is to classify wheter the input to-do belongs to the reminders list or the grocery list. To achieve this, an intent classification dataset from Hugging Face was used. The data looked like this:

Input	Label
"tell me how to set up a direct deposit"	direct_deposit
"switch to whisper mode"	whisper_mode
"i'm out of lysol could you order me some"	order
"set my alarm for 5pm"	alarm
"can you list my shopping list for me"	shopping_list
"update my shopping list, delete canned olives"	shopping_list_update
"can you list my shopping list for me"	shopping_list
"remove laundry from my todo list"	todo_list_update
"i want to hear everything on my todo list"	todo_list

The cagory classifier is trained on the whole dataset but only the labels that are relevant are used: shopping_list and shopping_list_update direct the flow to the grocery list, while todo_list and todo_list_update direct the flow to the to-do list. The category classifier is a scikit-learn Naive Bayes classifier and its code can be found in the subdirectory "src/py/todo_classifier".

In the subdirectory,

- `train_classifier.py`: is the training script that fetches the data from hugging face and then trains and exports the model as a raw python object with pickle. The dataseta was already partitioned by train/test/validation sets by hugging face.
- `use_classifier.py`: uses the saved model python object to label new input data.

Todo Extractor

The task of the Todo Extractor is to extract the task from the user's input. For example, if the user says "Add milk to my grocery list", the task is "Add milk". The Todo Extractor is implemented using OpenAI's pre-trained GPT 3.5 model. It was asked to output nice json objects that are much easier to work with programmatically. The prompt and the code can be found in the subdirectory "src/app/api/todoExtractor/".

Vision Food Detector

The task of the Vision Food Detector is to take a video stream and detect food items in it. The Vision Food Detector is implemented using a pre-trained YOLO model via Ultralytics and filtering the detected items to a set of labels containing only foods. The code can be found in the subdirectory "src/py/grocery_recognition/".

Grocery List Matcher

The task of the Grocery List Matcher is to take the items detected by the Vision Food Detector and cross them out from the grocery list. Like the Todo Extractor, the Grocery List Matcher is implemented using OpenAI's pre-trained GPT 3.5 model.

Initially the LLM model was asked to output the entire updated grocery list object, but empirically it made errors very often. Asking the LLM model for ids of the grocery list that were updated, and then updating the grocery list object programmatically, ended up working much better. The prompt and the code can be found in the subdirectory "src/app/api/groceryListMatcher/".

Query Agent

The task of the Query Agent is to elaborate queries. Once again, queries are inputs that don't carry new data within them. For example, "Split task X into subtasks", "What is the capital of Italy?.", or "Mark task Y as completed"

the Query Agent takes as input

- the voice input given by the user,
- the user's to-do list,
- and the user's grocery list

and gives as output

- a conversational 'narration' string to be given as audio feedback to the user,
- if asked to split an existing task into subtasks,
  - the category of the splitting task (to-do list or grocery list),
  - the id of the splitting task,
  - and the generated subtasks.
- and, if asked to mark a task as completed,
  - the category of the completed task (to-do list or grocery list),
  - the id of the completed task.

This agent as well gives instructions about what to change inside the lists rather than doing the change itself. The prompt and the code can be found in the subdirectory "src/app/api/conversationalAgent/".

Text-to-Speech Agent

To pronounce any generated narration string, the OpenAI pre-trained TTS model is used. Implementation at "src/app/api/tts".

Glue Orchestration

To make the whole app come together, the python tasks were made available to the network through a Python Flask server (code at "src/py/server.py"). A Node.js backend (code at "src/app/api/") makes requests to the Python server to use all the AI implementations, and finally the React font-end ("src/app/[username]/page.tsx") communicates with the Node.js backend to make everything available to the user through a user-friendly web user interface.

Results

The final result is a fully functioning proof of concept, but definitely not a product ready to be distributed to users. It works most of the time but it's not as reliable as should be.

Relying on the browser's API, Speech Recognition breaks in browsers that don't provide a working implementation. Doing this work in the backend would make the service more reliable and the app more versatile.

Even with 98% accuracy, the Question Classifier was still trained on a small dataset and a larger, more diversified dataset would make it more solid.

The category classifier was trained on separating "todo list" items from "shopping list" items, and shopping items are not always foods for the "grocery list". Training on a tailored dataset would make it more accurate.

All the GPT task are pretty good but rely on OpenAI's external model, which can increase delay in the waiting of the API call responses to get back to the machine from their servers, and, not being free, can become expensive over time.

The Vision Food Detector could be trained on custom images to recognize even more foods.