Aldone

Fausto Zamparelli, Daniel Falbo

May 30, 2024

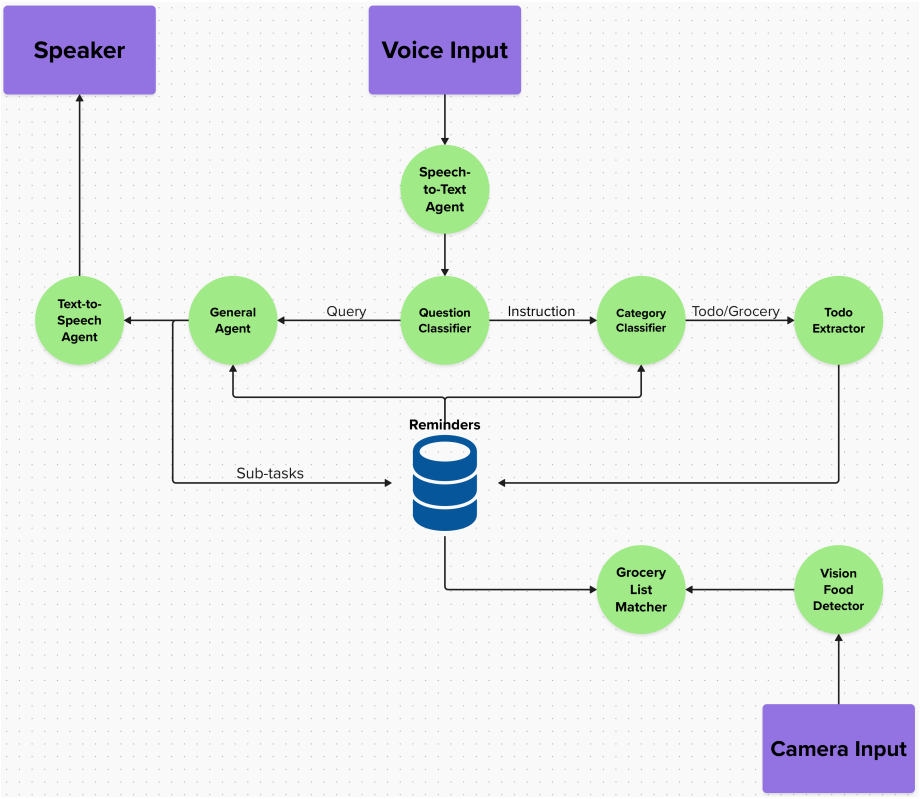
Introduction

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.

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. Of course it's still a small dataset and more data would make it more solid.

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.

The prompt used was

```
‘The following is a voice command recorded form
a voice assistant, extract the todo item from it to
insert it in a todo list.  For example, if the voice command
is 'Add 'Buy milk' to my shopping list', the extracted todo item
is 'Buy milk'.  You should also generate a "nice_readable_id"
and finally return a json object structured
exacly as {text: string; id: string}.
For example, {text: 'Buy milk', id: 'milk_buy'}.‘
```

This allowed to have nice json objects as output that are much easier to work with programmatically.

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.

The prompt used was

```
‘The following are the products that were
seen on the table: <products array>

The following is a json array of grocery list items
structured like [{text: string, completed: bool, id: string}].
Return me a json array of grocery list items ids of items that
were also seen on the table, if any.

Even if the ids don’t match exactly, you can still match
them if they are similar. For example, if the table has "apple"
and the grocery list has "red_apple", you can still match them.
Be smart and human about it. Make sure to return the id
as seen in the "id" field of the grocery list item,
not the products seen on the table.

The result should be formatted
like {ids: ["red_apples", "berries"]}.
If none of the items were seen on the table,
return an empty array "{ids: []}".

<grocery list object>‘
```

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 code can be found in the subdirectory “src/app/api/groceryListMatcher”.

General Agent

The task of the General 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" or "What is the capital of Italy?."

the General 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,
- and, if asked to split an existing task into subtasks,
  - the list of the splitting task (to-do list or grocery list),
  - the id of the splitting task,
  - and the generated subtasks.

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

Results

Glue Orchestration

Source of inspiration

[1] Cosmos: An Operating System for the AI Era
[2] aiXplain