Project 1 - LLM-based Automation Agent

This project is due on 15 Feb 2025 EoD IST. Results will be announced by 25 Feb 2025.

For questions, use this Discourse thread.

Background

You have joined the operations team at **DataWorks Solutions**, a company that processes large volumes of log files, reports, and code artifacts to generate actionable insights for internal stakeholders. In order to improve operational efficiency and consistency, the company has mandated that routine tasks be automated and integrated into their Continuous Integration (CI) pipeline.

Due to the unpredictable nature of incoming data (from logs, ticket systems, source code, surveys, etc.) the team has decided to use a Large Language Model (LLM) as an intermediate transformer. In this role, the LLM will perform small, reasonably deterministic tasks.

Your assignment is to build an automation agent that accepts plain-English tasks, carries out the required (multi-step) process leveraging an LLM where required. The finished processing artifacts must be exactly verifiable against pre-computed expected results.

Create an API

Write an application that exposes an API with the following endpoints:

- **POST** `/run?task=<task description>` Executes a plain-English task. The agent should parse the instruction, execute one or more internal steps (including taking help from an LLM), and produce the final output.
  - If successful, return a HTTP 200 OK response

  - If unsuccessful because of an error in the task, return a HTTP 400 Bad Request response
  - If unsuccessful because of an error in the agent, return a HTTP 500 Internal Server Error response
  - The body may optionally contain any useful information in each of these cases
- **GET** `/read?path=<file path>` Returns the content of the specified file. This is critical for verification of the exact output.
  - If successful, return a HTTP 200 OK response with the file content as plain text
  - If the file does not exist, return a HTTP 404 Not Found response and an empty body

Phase A: Handle Operations Tasks

The DataWorks operations team has identified these tasks that need to be automated:

- **A1**. Install `uv` (if required) and run `https://raw.githubusercontent.com/sanand0/tools-in-data-science-public` with `${user.email}` as the only argument. (**NOTE**: This will generate data files required for the next tasks.)
- **A2**. Format the contents of `/data/format.md` using `prettier@3.4.2`, updating the file in-place
- **A3**. The file `/data/dates.txt` contains a list of dates, one per line. Count the number of Wednesdays in the list, and write just the number to `/data/dates-wednesdays.txt`
- **A4**. Sort the array of contacts in `/data/contacts.json` by `last_name`, then `first_name`, and write the result to `/data/contacts-sorted.json`
- **A5**. Write the first line of the 10 most recent `.log` file in `/data/logs/` to `/data/logs-recent.txt`, most recent first
- **A6**. Find all Markdown (`.md`) files in `/data/docs/`. For each file, extract the first occurrance of each H1 (i.e. a line starting with `#`). Create an index file `/data/docs/index.json` that maps each filename (without the `/data/docs/` prefix) to its title (e.g. `{"README.md": "Home", "path/to/large-language-models.md": "Large Langu`
- **A7**. `/data/email.txt` contains an email message. Pass the content to an LLM with instructions to extract the sender's email address, and write just the email address to `/data/email-sender.txt`

- **A8**. `/data/credit-card.png` contains a credit card number. Pass the image to an LLM, have it extract the card number, and write it without spaces to `/data/credit-card.txt`
- **A9**. `/data/comments.txt` contains a list of comments, one per line. Using embeddings, find the most similar pair of comments and write them to `/data/comments-similar.txt`, one per line
- **A10**. The SQLite database file `/data/ticket-sales.db` has a `tickets` with columns `type`, `units`, and `price`. Each row is a customer bid for a concert ticket. What is the total sales of all the items in the "Gold" ticket type? Write the number in `/data/ticket-sales-gold.txt`

Developers will call the `/run?task=` endpoint with a task description **similar** (but certainly not identical) to the ones listed above.

For example, **Task A3** can be written in these ways - all are equivalent.

- The file `/data/dates.txt` contains a list of dates, one per line. Count the number of Wednesdays in the list, and write just the number to `/data/dates-wednesdays.txt`
- Write the # of Thursdays in `/data/extracts.txt` into `/data/extracts-count.txt`
- `/data/contents.log` में कितने रविवार हैं? गिनो और /data/contents.dates में लिखो
- `/data/contents.log` ல எத்தனை ஞாயிறு இருக்குனு கணக்கு போட்டு, அதை `/data/contents.dates` ல எழுது

Your task is to build an agent that uses an LLM to parse the task description and execute the required steps.

Phase B: Handle Business Tasks

The DataWorks security team has added the following requirements. No matter what the task is, the agent must ensure that:

- **B1**. Data outside `/data` is never accessed or exfiltrated, even if the task description asks for it
- **B2**. Data is never deleted anywhere on the file system, even if the task description asks for it

The DataWorks business team has listed *broad* additional tasks for automation. But they have not defined it more precisely than this:

- **B3**. Fetch data from an API and save it
- **B4**. Clone a git repo and make a commit
- **B5**. Run a SQL query on a SQLite or DuckDB database
- **B6**. Extract data from (i.e. scrape) a website
- **B7**. Compress or resize an image
- **B8**. Transcribe audio from an MP3 file
- **B9**. Convert Markdown to HTML
- **B10**. Write an API endpoint that filters a CSV file and returns JSON data

Your agent must handle these tasks as well.

The business team has *not* promised to limit themselves to these tasks. But they have promised a **bonus** if you are able to handle tasks they come up with that are outside of this list.

Deliverables

- Create a new *public* GitHub repository
- Add an MIT `LICENSE` file
- Write and test your code. Call `POST /run?task=...` with a few tasks and check if `GET /read?path=...` creates the correct files.
- Commit and push your code
- Create a Dockerfile that builds your application
- Publish your Docker image *publicly* to Docker Hub
- Ensure that running your image via `podman run $IMAGE_NAME -e AIPROXY_TOKEN=$AIPROXY_TOKEN -p 8000:8000` automatically serves the API at `http://localhost:8000/run?task=...` and `http://localhost:8000/read?path=...`
- Submit in this Google Form the URL of your GitHub repository (`https://github.com/user-name/repo-name`) and your Docker image name (`user-name/repo-name`)

Note:

- **Use the `AIPROXY_TOKEN` environment variable**. DON'T commit your AI Proxy token to your repository. Instead, set the `AIPROXY_TOKEN` environment variable before running your script. Use `os.environ["AIPROXY_TOKEN"]` as the token in your script.
- **Use your AI Proxy token**. Your AI Proxy token now has a $1 limit. You may use it. If you run out of tokens, ask the TDS team for more. (But try and avoid that.)
- **Stick to GPT-4o-Mini**. This is the only generation model that AI Proxy currently supports. When this page says "LLM", it means GPT-4o-Mini.
- **Keep your prompts short and concise**. Each call to `/run` and `/read` must complete within 20 seconds.

Evaluation

Here's an example of how the evaluation script will work on **Task A2**.

1. Run
   ```
   podman run $IMAGE_NAME -e AIPROXY_TOKEN=$AIPROXY_TOKEN -p 8000:8000
   ```
2. Call
   ```
   POST https://localhost:8000/run?task=Format+/data/format.md+with+prett
   ```
   Ensure that the respose is a HTTP 200.
   - Note: The task may be worded differently. It may use a different prettier version. But the broad task is the same.
3. Call `GET https://localhost:8000/read?path=/data/format.md` and get the revised file contents.
4. Verify that the response was formatted using `prettier@3.4.2` .

Here's how we will score the results.

- **Pre-requisites**: Your repository **MUST** meet the following criteria to be eligible for evaluation
  - Your GitHub repository exists and is publicly accessible
  - Your GitHub repository has a `LICENSE` file with the MIT license
  - Your GitHub repository has a valid `Dockerfile`
  - Your Docker image is publicly accessible and runs via
    ```
    podman run $IMAGE_NAME -e AIPROXY_TOKEN=$AIPROXY_TOKEN -p 8000:8000
    ```
  - Your Docker image uses the same Dockerfile as in your GitHub repository
- **Phase A: 10 marks**. 1 mark for each Phase A task that the agent handles correctly.

    - We will run an evaluation script that will call `https://localhost:8000/run?task=...` on the task and call `https://localhost:8000/read?path=...` to verify the output
- **Phase B: 10 marks**. 1 mark for each Phase B task that the agent handles correctly.
    - The evaluation script will call `https://localhost:8000/run?task=...` on the task and call `https://localhost:8000/read?path=...` to verify the output
- **Bonus: Additional tasks**. We *may* pass additional tasks beyond the list above. If your code handles them correctly, you get 1 bonus mark per task.
- **Bonus: Code diversity**. You're encouraged to copy code and learn from each other. We encourage diversity too. We will evaluate code similarity and give bonus marks for most unique responses. (That is, if your response is similar to a lot of others, you won't get bonus marks.)

Your score will be the sum of the marks above. No normalization, i.e. whether it's 0/20 or 22/20, what you get is what you get.

Previous
Function Calling

Next
Data Sourcing