Micro-Immersion GenAI

More than just chitchat

Summary: Integrate AI into your apps to tackle real-world problems.

Version: 1.0

Contents

Chapter I

Foreword

Stochastic parrot

In the field of machine learning, the phrase 'stochastic parrot' serves as a metaphor to illustrate the idea that, although large language models can produce coherent language, they lack an understanding of the meaning behind the language they generate. This term was introduced by Emily M. Bender in the 2021 AI research paper titled 'On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? :parrot:' authored by Bender, Timnit Gebru, Angelina McMillan-Major, and Margaret Mitchell.[1]

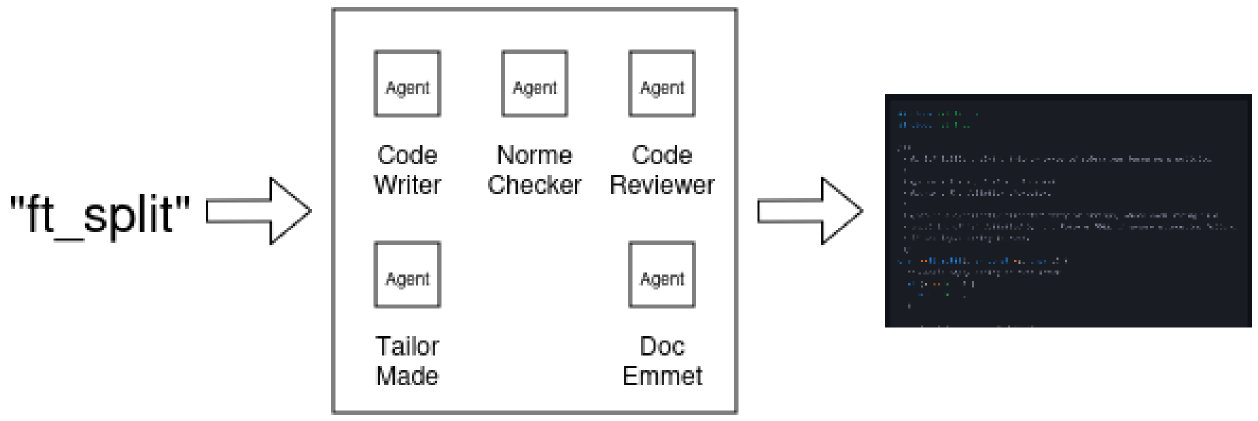
[1] https://en.wikipedia.org/wiki/Stochastic_parrot

Chapter II

Introduction

In this project, you'll be building a multi-agent system[1] using LLMs (Large Language Models).

The system is composed of several agents that collaborate to create C functions based on input parameters. Each agent has a specific role and contributes to refining the code and generating the necessary outputs. The system will look something like this:



- Code Writer: This agent is responsible for writing the code based on the data it receives.

- Norme Checker: This agent ensures that the code complies to the Norminette coding standards, guaranteeing compliance with predefined coding rules.

- Code Reviewer: This agent generates questions to help you prepare for the evaluation that will occur when you submit your code for review.

- Tailormade: This agent reviews your code on GitHub to understand your coding

[1]https://en.wikipedia.org/wiki/Multi-agent_system

style and ensures that the generated code closely resembles your own style.

- Doc Emmet: This agent generates documentation for your functions in Markdown format.

This system will occasionally need to search the internet and local documents to ensure the best output, and also seek human feedback for further improvement. Everything will be encapsulated in a user-friendly frontend where you can manage these interactions.

Chapter III

General instructions

- This project will be reviewed by humans.
- For this project, you must use Python.
    - Python version >= 3.8
    - Use a virtual environment (venv) in your project. It's also good to install the dependencies in your venv.
- There are several libraries for interacting with LLMs in the Python ecosystem, with LangChain[1] being the most popular currently.
    - We will use CrewAI[2] as the primary library.
    - We will use LangChain (CrewAI uses it under the hood) but only minimally.
    - For the frontend, we will use Streamlit[3], the most popular library for frontend systems leveraging AI.
- As for models, we will use Google's models.
    - We will stick with Google AI Studio[4], which gives you a fair enough token per minute quota for `free` using a Google account.
    - We recommend using `gemini-1.5-flash` for the highest quota. You can try using different models for various agents.
    - Local models can be an option, but they will run very slowly without a GPU. You can take a look at Ollama[5].
- As usual, there should be no errors or warnings in any console. HTTP without HTTPS browser warning is not considered an issue.

[1]https://www.langchain.com/
[2]https://www.crewai.com/
[3]https://streamlit.io/
[4]https://ai.google.dev/aistudio
[5]https://ollama.com/

Chapter IV

Mandatory part

IV.1 Common features

- The system should receive an input, which is the name of the function to generate.
    - You can add more inputs to make the request more detailed, but this one is mandatory.
- The output should:
    - Be the requested function.
    - Follow the correct standards defined by each agent's role (see below).
    - Fulfill the C function properly.
    - The best output possible by using human feedback and various types of search as needed.
    - Include the corresponding documentation.
- The system should be composed of multiple agentic LLMs, each with a well-defined task:
    - Code Writer: This agent will gather all necessary information and write the code.
    - Code Reviewer: This agent will review the code and generate questions about it, simulating a classic evaluator.
    - Norme Checker: This agent ensures the code complies with the 42 coding guide.
    - Tailormade: This agent reviews your code on GitHub to understand your coding style and ensures the generated code closely matches your own style.
    - Doc Emmet: This agent generates documentation for your function in Markdown format, which can then be incorporated into a global documentation system.
- The frontend should include the following features:
    - At least one input field.
    - A button to start the process.

  - A section to display the generated output.
  - Configuration options for entering the API key for the LLM.
  - A field to input a GitHub profile URL to be scraped by the Tailormade agent.
  - Options to define which set of models to run (e.g., Gemini for Google, GPT-4 for OpenAI).
  - A method to input human feedback if needed.
  - Options to point to local documents.
  - A toggle to enable or disable internet search.

- Ensure the frontend is user-friendly and intuitive, making it easy to understand what's happening. Including explanations of the outputs is recommended, for example.
- Make sure to handle re-requests/retries correctly so your app does not crash or stop responding. If it cannot provide an answer, inform the user.
- Provide a way to call your system via the CLI, allowing direct use through the terminal, such as: `python llm.py ft_split`.
- Include a `setup.sh` file that will install and configure everything so the app can run correctly.

Chapter V

Turn-in and peer-evaluation

V.1 Turn-in

Submit your work to the GitHub repository generated when you accepted the project invitation.

V.2 Peer-evaluation

Someone from Bocal will evaluate your project and code, assessing the quality of your work.