AI Agents in Action

Transforming Software Development

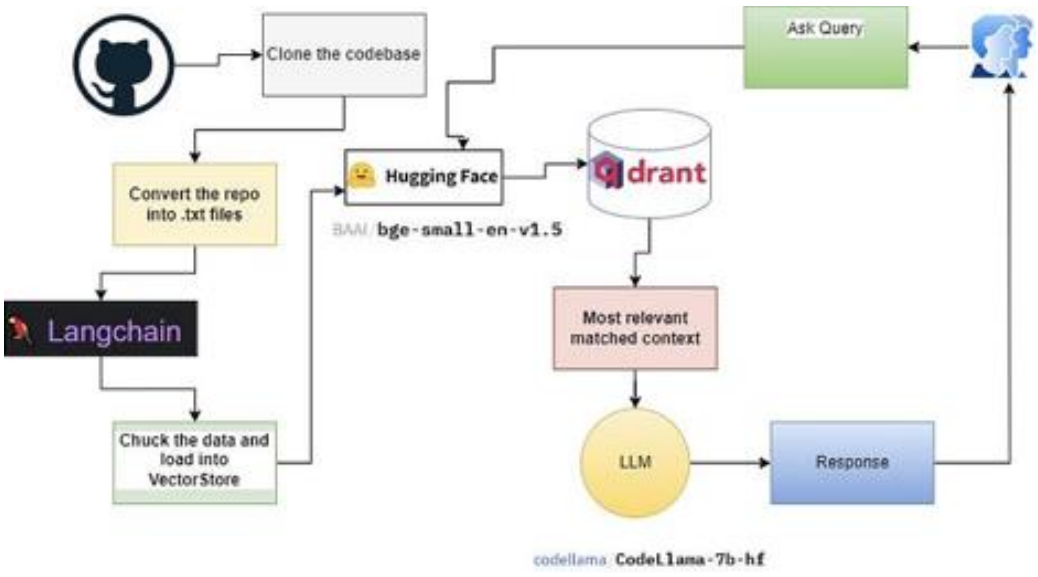
1. The Challenge

Objective

Our customer was contacting us concerning an existing software development project. They do currently have the problem of not having enough developers to cope with the requirements given to the development department within the given time. Thereby they want us to support their developers with GenAI agents. The idea is to reduce the workload and improve the quality of the developments.

Your task thereby will be to develop a GenAI Agent, that supports the developers of our customer, based on a Retrieval-Augmented Generation (RAG) application that allows users to query an open-source codebase and receive contextually relevant answers generated by a large language model (LLM), grounded in the actual code.

An example architecture is as follows:



Our Expectations:

- **Develop** a concrete **software-engineering task** that could be improved with Retrieval-Augmented Generation (RAG). Define the process of how the agent will support.
- **Assemble** a mini corpus (code, docs, logs, etc.) that grounds your solution.
- **Build** an *automated* pipeline—one command or endpoint turns an input into a helpful answer/action.
- **Demonstrate** it live and hand in a repo + README that anyone can run
    - Prepare 1 slide for a max 6 min presentation.

2. Challenge Evaluation

Success Criteria

- **Originality:** Have a unique idea / approach and be able to share your brilliance
- **Correctness**: Relevant code context is retrieved and used in answers.
- **Responsiveness**: Answers are generated within a reasonable time.

Must-Have Requirements

1. Problem Description and Use Cases:
2. **Retrieval:** At least one vector search over *your* corpus.
3. **Generation:** An LLM prompt that references retrieved chunks (citations or copied text).
4. **Automation:**
   • accepts the user input
   • runs retrieval & generation
   • prints/returns the answer.

Nice-to-Have Extras (for tie-break points)

- Multiple agent approach to solve more and potentially interacting software engineering tasks.
- Re-ranker, Hybrid Score or multi-stage retrieval.

- Caches / batching for speed.
- Guard-rails against hallucination.
  CI-ready: pipeline exits non-zero on failure, prints JSON.

3. Technical requirements

Tools & Technologies

- **LangChain** – Framework for LLM applications
- **Qdrant** – Vector database for similarity search
- **Hugging Face Transformers** – Pretrained models (e.g., CodeLlama-7b)
- **Gradio** – Web UI interface for interaction
- **Sentence Transformers** – For embedding generation
- Other tools of your preference

Dataset

- Clone any open-source GitHub repository (e.g., chat-with-websites)
- Focus on file types: `.py`, `.md`, `.txt`

Implementation Steps

1. **Environment Setup**
   Install required packages:
   ```
   pip install langchain transformers accelerate
   sentence-transformers qdrant-client langchainhub
   gradio
   Install additional packages if required.
   ```
2. **Clone the Repository**
   Use Git to clone the target codebase:
   `git clone https://github.com/alejandro-ao/chat-with-websites codebase`
3. **Document Loading**
   Parse the repository and extract relevant text from supported file types.

4. **RAG Pipeline**:
   Follow the steps of the basic RAG demonstrated in the workshop. Implement your own RAG with the given repository.