Flint

iClaudes - Giorgia Savo, Marzia Favitta, Leonardo Sinibaldi, Simone Ranfoni

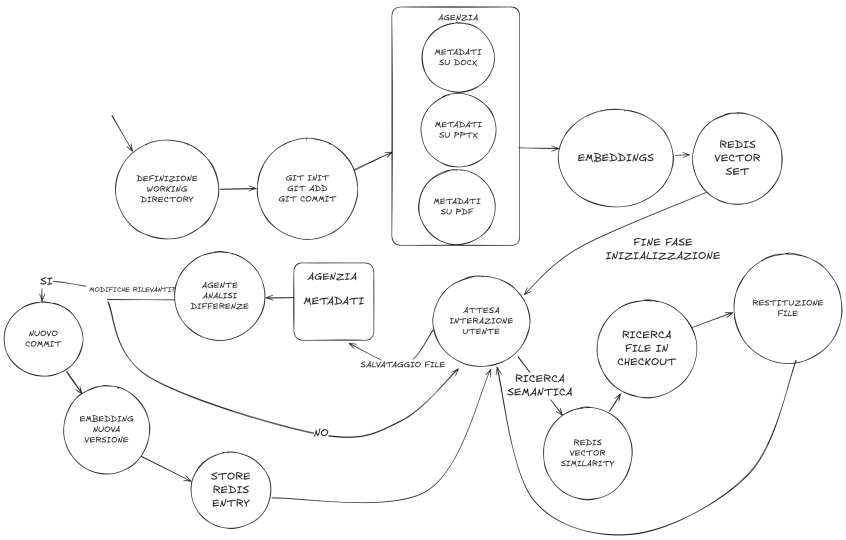
May 11, 2025

1 Introduction

Managing and retrieving files within systems handling vast amounts of data can prove complex. Users frequently recall fragments of a file's content but struggle to locate it using traditional search methods reliant on filenames or basic keywords. This project addresses this challenge by introducing an AI agent that (1) enables semantic file search and (2) automatically tracks file versions, facilitating content-based 'temporal' searches. The objective is to provide a more intuitive and efficient human-computer interface for information navigation and retrieval, thereby eliminating redundant clicks.

2 System Architecture

The system comprises an Electron frontend application and a Python backend, communicating via Sockets. Redis is employed for storing file embeddings and metadata, while Git manages low-level version control.



2.1 Frontend (Electron)

The frontend is built using Electron, providing a cross-platform user interface akin to a 'spotlight search' window.

- **main.js**: The main Electron process. It manages the application lifecycle, window creation (BrowserWindow), registration of global shortcuts (e.g., Option+Space to show/hide the window), and IPC communication with the renderer process. It initializes the Socket connection to the backend.
- **renderer.js**: Logic for the renderer process (UI). It handles user input, displays search results, and manages interaction with them (e.g., opening files). It communicates with the main process via the API exposed by `preload.js`.
- **preload.js**: A script that securely exposes selected Electron functionalities (like `ipcRenderer`) to the renderer process, maintaining context isolation.

2.2 Backend (Python)

The backend handles the core business logic, including file processing, embedding generation, search, and versioning.

- **app.py**: The central component of the backend.
  - Initializes the connection to Redis and loads the embedding model (SentenceTransformer).
  - Uses `watchdog` to monitor file system changes (creation, modification, deletion, moves) within a specified directory.
  - Manages a Git repository in the monitored directory, automatically creating commits for each relevant file modification (primarily .docx files).
  - For each modified and committed file, it invokes `agent.py` to extract text and generate a summary.
  - Generates vector embeddings from the file summary using the SentenceTransformer model.
  - Stores file metadata (name, relative path, commit hash, summary) and its vector embedding in Redis. The record ID in Redis combines the file's relative path and the commit hash, enabling the indexing of different versions of the same file.
  - Starts a Socket server (implemented in `ChangeHandler`) to listen for search requests from the frontend.
- **agent.py**: Responsible for extracting textual content from various file types (DOCX, PDF, XLSX, PPTX) and generating meaningful summaries using a generative language model (Google Gemini). The summaries are designed to capture the document's essence and include keywords.
- **searcher.py**: Receives the user query from the frontend (via `app.py`).
  - Utilizes Google Gemini to analyze and "expand" the user's query, transforming it into a hypothetical description of the sought document's content. This enriched text is then used for semantic search.
  - Generates an embedding for the analyzed query.
  - Performs a vector similarity search in Redis (against the indexed file embeddings) to find the most relevant files/versions.
  - Returns the paths of the corresponding files to the frontend.
- **diff.py**: Provides functionality to compare two versions of a document. It uses Google Gemini to analyze a summary of the current version and a summary of a previous version (retrieved from Redis) and determines if the changes are "significant", returning 0 (similar) or 1 (significantly different).

2.3 Storage and Versioning

- **Redis**: Used as a vector database to store file summary embeddings and as a key-value store for associated metadata (path, commit hash, textual summary). The key for each entry includes the commit hash, enabling the indexing and retrieval of specific versions.
- **Git**: Integrated into `app.py` to automatically create commits for file changes within the monitored directory. This provides the foundation for version tracking. The Git commit hash is a fundamental part of the unique identifier for a file version in Redis.

3 Technologies Used

- **Python**: For backend development.
- **JavaScript (Node.js)**: For frontend development with Electron.
- **Electron**: Framework for building cross-platform desktop applications using web technologies.
- **Redis (with Redis Stack/Search)**: NoSQL database used for metadata storage and as a vector database for similarity search.
- **Git**: Distributed version control system, used for file versioning.
- **Google Gemini**: Generative AI model used for query analysis, summary generation, and version comparison.
- **SentenceTransformers**: Python library for generating text embeddings.
- **Watchdog**: Python library for monitoring file system events.
- **Socket.IO / Sockets**: For real-time communication between the frontend and backend.