Features

Table of contents

1 Features

This page provides a detailed overview of the features available in `code_agent`.

1.1 Core Features

`code_agent` is built around a few core concepts that make it a powerful and flexible tool for local development.

1.1.1 1. Project Scaffolding

Start new Python projects in seconds with the `scaffold` command. `code_agent` generates a complete, modern project structure, so you can start coding right away.

The generated scaffold includes: - A `src` directory for your package source code. - A `tests` directory with a sample smoke test. - A `docs` directory ready for Quarto documentation. - A pre-configured `.github/workflows/ci.yml` for continuous integration. - Standard configuration files like `requirements.txt` and `README.qmd`.

1.1.2 2. File Generation and Manipulation

`code_agent` provides a rich set of utilities for programmatic file I/O, accessible via the Python API or the interactive agent. - **Create and Write Files**: Easily create new files or overwrite existing ones. - **Append Content**: Add content to the end of existing files. - **Template-Based Generation**: Create files from templates with variable substitution. - **Python to Notebook Conversion**: The `py2ipynb` CLI command converts Python scripts with `#%%` cell markers into Jupyter notebooks.

1.1.3 3. LLM-Powered Interactive Agent

The heart of `code_agent` is its intelligent agent, which uses a local LLM (via Ollama by default) to understand and respond to natural language commands. The agent can: - **Generate and Explain Code**: Create new functions, classes, or entire modules, and explain how they work. - **Automate Testing**: Generate `pytest` test files for your source code. - **Orchestrate Tools**: Combine its tools to perform complex, multistep tasks based on your requests. - **Maintain Context**: The agent remembers previous turns in the conversation, allowing for iterative development workflows.

1.1.4 4. Extensible Toolset

The agent's capabilities are defined by its tools. `code_agent` comes with a powerful set of built-in tools for file I/O, code formatting, and analysis. It's designed to be easily extended with your own custom tools. See the Tools** page for a full list and instructions on how to create your own.

1.2 Architecture Overview

`code_agent` has a modular architecture built on top of LangChain and LangGraph.

```
graph TD;
   subgraph "User Interface"
       CLI[code_agent/cli.py]
       InteractiveLoop[code_agent/main.py]
   end

   subgraph "Agent Core"
       Graph[code_agent/graph.py]
       AgentFactory[code_agent/agents/base_agent.py]
       Tools[code_agent/tools/]
```

```
    end

    subgraph "LLM & Memory"
        LLM[LLM Wrapper]
        VectorStore[ChromaDB]
    end

    subgraph "Core Utilities"
        FileOps[code_agent/file_generator.py]
        Scaffolding[code_agent/scaffold.py]
    end

    CLI --> InteractiveLoop;
    InteractiveLoop --> AgentFactory;
    AgentFactory --> Graph;
    Graph --> Tools;
    Graph --> LLM;
    Graph --> VectorStore;
    Tools --> FileOps;
    CLI --> Scaffolding;
```

Key Components:

- **User Interface**: Provides both a command-line interface (`cli.py`) for direct commands and an interactive chat loop (`main.py`).
- **Agent Core**: The central component that defines the agent's logic. `base_agent.py` creates the tools, and `graph.py` orchestrates the flow of control between the LLM and the tools.
- **LLM & Memory**: The agent uses an LLM (e.g., `ChatOllama`) for reasoning and a `Chroma` vector store to maintain conversational memory.
- **Core Utilities**: A set of modules that provide the fundamental building blocks, such as file I/O and project scaffolding.