Roadmap

Project Roadmap

This document outlines the future direction for `code-agent`, focusing on enhancing its capabilities, improving user experience, and expanding its toolset.

Core Philosophy

The development of `code-agent` is guided by these principles:

1. **Local First**: All core functionality must run on a local machine without reliance on external, paid APIs. User privacy and data security are paramount.
2. **Extensibility**: The agent should be easy to extend with new tools and capabilities.
3. **Practicality**: Tools and features should solve real-world development problems and integrate smoothly into existing workflows.
4. **Backend Agnostic**: While `Ollama` is the default, the architecture should remain flexible enough to accommodate other local LLM providers.

Short-Term Goals (Next 1-3 Months)

1. Enhanced Codebase Awareness (Advanced RAG)

- **Goal**: Improve the agent's ability to understand the entire project structure.
- **Tasks**:
  - Implement a file indexer that runs in the background to create embeddings for all source code files.
  - Use Retrieval-Augmented Generation (RAG) to inject relevant code snippets from the entire codebase into the prompt, not just from conversation history.
  - Develop a "Codebase Q&A" tool that can answer questions like "Where is the database configuration handled?"

2. Improved Toolset

- **Goal**: Make the existing tools more robust and add new, high-value tools.
- **Tasks**:
  - **Git Integration**: Create a `git` tool that can perform operations like `git diff`, `git status`, and `git commit`.
  - **Dependency Management Tool**: A tool to add, remove, or update dependencies in `pyproject.toml` or `requirements.txt`.
  - **Refactoring Tool**: A more advanced version of `edit-file` that can perform structured refactoring, such as renaming a function and updating all its call sites.

3. Better User Experience

- **Goal**: Make the agent easier and more intuitive to use.
- **Tasks**:
  - **Interactive Mode Improvements**: Add features like tab-completion for commands and better multi-line input handling in the chat interface.
  - **Streaming Output**: Ensure that all tool outputs and LLM "thinking" steps are streamed back to the user in real-time for better transparency.
  - **Configuration Wizard**: A command to help users set up their `llm_config.json` for the first time.

Medium-Term Goals (3-6 Months)

1. Multi-File Operations

- **Goal**: Enable the agent to perform tasks that involve changes across multiple files in a single, atomic operation.
- **Tasks**:
  - Develop a "transaction" or "workspace" concept where the agent can stage changes to multiple files.
  - Add `apply` and `discard` commands to either commit the staged changes or throw them away.
  - Ensure that multi-file changes are presented to the user as a single, unified diff for review.

2. Testing and CI/CD Integration

- **Goal**: Make the agent an active participant in the testing and deployment lifecycle.
- **Tasks**:
  - **Test Execution and Debugging**: A tool that can not only run `pytest` but also interpret the results and attempt to fix failing tests.
  - **CI/CD Helper**: A tool that can help generate or update GitHub Actions workflows.

3. Provider Flexibility

- **Goal**: Officially support more local LLM providers.
- **Tasks**:
  - Create a unified provider interface.
  - Add tested integrations for other popular local LLM servers like `LMStudio` and `vLLM`.

Long-Term Vision (6+ Months)

- **Graphical User Interface**: A simple desktop application (e.g., using Qt or a web-based UI) to provide a richer user experience than the command line.
- **Multi-Modal Support**: The ability to process not just text but also images (e.g., architectural diagrams) as input.
- **Proactive Agent**: An agent that can run in the background, monitor your codebase for potential issues (like code smells or dependency vulnerabilities), and proactively suggest improvements.

We welcome contributions! If you are interested in working on any of these features, please see our CONTRIBUTING.md guidelines.