Helix CLI Specs

The Helix CLI is the AI coding agent built for the terminal to access and work (code, test, debug, etc.) with LLMs. Primary provider is LLama CPP, however it supports all other options such as work with remote APIs - DeepSeek, Qwen, Claude, Gemini, Grok, Mistral, etc. It supports various other providers like OpenRouter, Ollama, Nvidia, HuggingFace, etc.

It is powerfull programming tool which intracts with end user and makes possible flawless code implementation on the projects, refactopring, debugging, etc.

Important power feature

The CLI makes possible for end users to build and run local LLMs and use them for code generation, testing, debugging, etc. However, most important is that Toolking, Thinking and other LLM capabilities may be flwalessly used on low end computer systems (by using 7B LLMs). On more capable machines, dynamically detrmined, more powerful LLMs may be used - 32b and similar.

Architecture

All project / application components are fully decoupled programs which are bound together with proper bash scripts. Each comonent can be used separately or as a part of larger application bound together by bash scripts. All programs are writtent in Go language.

Some of the components that we are mentioning now are:

- LLama CPP API for communication with LLMs executed locally by LLama CPP
- OpenRouter API for communication with OpenRouter
- Ollama API for communication with Ollama
- DeepSeek API for communication with DeepSeek
- Qwen API for communication with Qwen
- Claude API for communication with Claude
- Other APIs for communication with other LLMs
- ASII art generator - will generate the header for the project that will appear in CLI UI when user starts the application
- Pure CLI mode (Headless CLI mode)
- CLI UI mode
- REST API mode (run as server)
- Postgres database component (run as server) - will be used by CLI UI and REST API, storing memories, progres, etc.
- Debugger component - make sure that debugging of software for issues detection is possible using LLMs. It investigates and collects information about software execution for further fixes and improvements or new features development.
- Planner - Does the deep analysis and perfroms the planning with written documentation as result of this efforts - user inputs are required as always
- Builder - Builds the code using LLMs, extends the features, fixes issues, etc.
- Refactorer - Refactors the code using LLMs, extends the features, fixes issues, etc.

- Tester - Tests the code using LLMs, extends the features, fixes issues, etc. Tester introduces to the codebase of single or multiple modules of the projects the full 100% code coverage with several testing types: unit, integration, full automation, end-to-end, etc. Last two types will always be executed on real devices or emulators or apps ran by real AI QA! There are two more kind of tests that will always be performed: SoanrQube deep scanning and Snyk. Both performed with dockerized (docker compose) free versions of the two tools. Full reports will be generated and any discovered issues fully fixed.

LLama CPP and Ollama support

Under the hood API will contain its local instance (codebase) of LLama CPP and Ollama. It will be used for communication with LLMs via exposed API. Once it is ready it will be used for communication with LLMs. To be ready it has to download source code, do optimal build for current OS, do the configuration and optimization. API will expose downloadable LLMs for both LLama CPP and Ollama. It will download LLMs from all possible sources such as HuggingFace, OpenRouter, DeepSeek, Qwen, Claude, etc. Downloaded LLMs will be converted to proper format if that is needed so LLama CPP and Ollama will be able to use them. User will trigger the installation of the LLM and it will be ready for use. Installation will first make sure to detect hardware of host machine - all mandatory capabilites, then it will build local project instance of LLama CPP and Ollama and will download the LLM to be used. All optimization performed and LLM downloaded and started.

All utils have to be decoupled and other components for installation, configuration, download and running the LLMs. Make sure we have proper generics and interfaces which are widely reusable! Resusability principle has to be followed with every project component!

Others APIs - VLLM, Local LLM, etc

Same principle we described for LLama CPP and Ollama will be applied to other platforms and their LLMs.

CLI UI features

CLI (Headless) features

REST API features

Testing

Tbd