Sqirvy-cli

Sqirvy-cli is a versatile command-line interface (CLI) tool designed for interacting with various Large Language Models (LLMs). It offers implementations in both Go and Python, providing a consistent experience across different development environments.

The tool allows users to leverage the power of multiple AI providers directly from their terminal, integrating smoothly into development workflows and scripting pipelines.

Implementations

This repository contains two primary implementations:

1. **Go (`go/`)**: A native executable built using Go. It utilizes libraries like Cobra for the CLI structure, Viper for configuration, `langchaingo` for LLM interactions, and `colly` for web scraping.
2. **Python (`python/`)**: A standard Python package. It uses `argparse` for command-line argument parsing, the `langchain` ecosystem for LLM interactions, and `requests`/`beautifulsoup4` for web scraping.

Key Features (Common to both Go and Python)

- **Multi-Provider Support**: Interact with models from major providers:
  - Anthropic (Claude models)
  - Google (Gemini models)
  - OpenAI (GPT models)
  - Llama (via OpenAI-compatible APIs)
- **Command-Driven Interface**: Offers distinct commands tailored for specific tasks:
  - `query`: Send arbitrary prompts or questions to the selected LLM. (Default command if none is specified).
  - `plan`: Request the LLM to generate a plan or design based on the provided input.
  - `code`: Ask the LLM to generate source code based on a prompt or plan.
  - `review`: Instruct the LLM to perform a review of the provided code or text.
  - `models` (Go only, Python shows via help): List the supported LLM models and their corresponding providers.
- **Flexible Input Sources**: Accepts input prompts through multiple channels:
  - **Standard Input (stdin)**: Enables seamless integration with Unix pipelines (e.g., `cat file.txt | sqirvy-cli ...`).
  - **File Paths**: Directly process the content of local files.
  - **URLs**: Automatically scrape and use the text content from web pages.
- **Configuration**:
  - **Model Selection**: Specify the desired LLM using the `-m`/`--model` flag. Model aliases are supported (e.g., `claude-3-opus` maps to `claude-3-opus-latest`).
  - **Temperature Control**: Adjust the creativity/randomness of the LLM's output using the `-t`/`--temperature` flag (typically 0.0 to 1.0 or 2.0 depending on the provider's scale, the tool handles scaling internally).
  - **API Credentials**: Configure API keys and necessary base URLs via environment variables

    - ANTHROPIC_API_KEY
    - GEMINI_API_KEY
    - LLAMA_API_KEY
    - LLAMA_BASE_URL
    - OPENAI_API_KEY
    - OPENAI_BASE_URL

These variables are necessary for the respective clients (Anthropic, Gemini, Llama, OpenAI) in both the Go and Python implementations to authenticate and connect to the corresponding LLM provider APIs.

- **Structured Interaction**: Uses predefined system prompts for each command (`plan`, `code`, `review`, `query`) to provide context and guide the LLM towards the desired output format and task execution.
- **Modular Design**: Both implementations separate the user-facing CLI logic from the core LLM interaction library (`pkg/sqirvy` in Go, `sqirvy_cli/sqirvy` in Python), promoting maintainability and reusability.

Getting Started

Refer to the `README.md` files within the `go/` and `python/` directories for specific build, installation, and usage instructions for each implementation.

Ensure you have the necessary API keys set as environment variables for the providers you intend to use.