Sqirvy-cli (Python Implementation)

This directory contains the Python implementation of `sqirvy-cli`, a command-line tool for interacting with various Large Language Models (LLMs).

Overview

The Python version of `sqirvy-cli` provides a flexible and extensible way to query LLMs from the terminal using the popular `langchain` ecosystem. It is packaged as a standard Python CLI tool installable via `pip`.

Key Features

- **Python Package**: Built as a standard Python package using `setuptools` and `pyproject.toml`.
- **Multi-Provider Support**: Interacts with Anthropic, Google Gemini, OpenAI, and Llama models via the `langchain` library (`langchain-anthropic`, `langchain-google-genai`, `langchain-openai`).
- **Structured Commands**: Uses the `argparse` library for command-line argument parsing:
  - `query`: Sends arbitrary prompts (default command if none specified).
  - `plan`: Requests the LLM to generate a plan.
  - `code`: Asks the LLM to generate source code.
  - `review`: Instructs the LLM to review code or text.
  - Model listing is integrated into the `--help` output.
- **Flexible Input**: Reads prompts from:
  - Standard Input (stdin) for easy piping.
  - File paths.
  - URLs (content is scraped using `requests` and `beautifulsoup4`).
- **Configuration**:
  - Command-line flags (`-m`/`--model`, `-t`/`--temperature`) managed by `argparse`.
  - Environment variables for API keys (`ANTHROPIC_API_KEY`, `GEMINI_API_KEY`, `OPENAI_API_KEY`, `LLAMA_API_KEY`) and base URLs (`OPENAI_BASE_URL`, `LLAMA_BASE_URL`) accessed via helper functions in `sqirvy/env.py`.
- **System Prompts**: Uses predefined prompt strings defined in `sqirvy/prompts.py` for each command.
- **Modular Design**:
  - `sqirvy_cli/main.py`: Main entry point, handles argument parsing and orchestrates execution.
  - `sqirvy_cli/sqirvy/`: Core library containing the LLM interaction logic.
    - `client.py`: Defines the abstract `Client` base class and the `new_client` factory function.
    - `context.py`: Defines the `Context` dataclass to hold all execution parameters and handles input aggregation.
    - `models.py`: Manages model-to-provider mapping, aliases, and token limits.
    - `prompts.py`: Contains the system prompts for different commands.
    - `env.py`: Handles environment variable retrieval.
    - `query.py`: Provides a common `query_text_langchain` helper function.

        - `*_client.py`: Provider-specific client implementations (Anthropic, Gemini, OpenAI, Llama) inheriting from `Client`.
    - `sqirvy_cli/utils/`: Contains utility functions.
        - `files.py`: Implements file reading and URL scraping.

Installation

1. **Prerequisites**: Ensure you have Python 3.8+ and `pip` installed.
2. **Install Dependencies**: Navigate to the `python/` directory and install the required packages:

```
cd python
pip install -r requirements.txt
```

3. **Install the CLI tool**: Install the `sqirvy-cli` package itself:

```
pip install ./sqirvy_cli
```

Alternatively, for development, install in editable mode:

```
pip install -e ./sqirvy_cli
```

Running

Once installed, you can run the tool from your terminal using the `sqirvy_cli` command:

```
# Basic query using a specified model
# (Ensure required API key env var is set, e.g., OPENAI_API_KEY)
echo "What is the airspeed velocity of an unladen swallow?" | sqirvy_cli -m
gpt-4o

# Specify model and temperature, providing a file
sqirvy_cli -m claude-3-5-sonnet-latest -t 0.7 -c query my_prompt.txt

# Generate a plan from stdin (using default temperature 1.0)
cat requirements.txt | sqirvy_cli -c plan -m gpt-4o

# Generate code based on a plan file and a URL
sqirvy_cli -c code -m gemini-1.5-pro plan.md https://example.com/api-spec

# Show help, including available models
sqirvy_cli --help
```

Remember to set the required API key environment variables for the models you intend to use.

Testing

Unit tests are included and use Python's built-in `unittest` framework. You can run tests using the provided shell scripts or standard discovery:

```
# Run all tests using Python's discovery
cd python/sqirvy_cli/sqirvy_cli
python -m unittest discover .

# Or use the provided test scripts (example)
./test_cli.py # May require adjustments based on script content/permissions
```

Note: Some tests might require API keys to be set in the environment and may interact with live services.