Westlake University

LLM Github Installer

Author:
Haotian Shen
Xun Zhang
Zixuan Wang

Final Report for NLP Project

June 7, 2025

1 Overview

In this project, we developed "LLM Github Installer", an intelligent LLM-based command-line tool designed to simplify the setup and deployment of open-source projects hosted on GitHub. Leveraging the power of Large Language Models (LLMs), the tool automates the process of analyzing a project's README file and generating the necessary installation and configuration commands. We want this tool to help users quickly and efficiently set up complex development environments with minimal manual intervention.

The core components of the project include:

- Intelligent Analysis: Utilizes LLMs API like Alibaba's Qwen and Google's Gemini to parse and understand project documentation.
- Automatic Command Generation: Creates a sequence of installation commands tailored to the specific project and the user's system.
- Interactive Execution: Provides an interactive terminal where users can execute, skip, or edit the suggested commands one by one.
- Error Handling: Feeds execution errors back to the LLM to generate corrective commands.
- User-Friendly Interface: we enable the user to add prompt during commands generation.Also user can choose to run/skip/edit/quit during the process.

2 Method and Technique

The project's workflow is designed to be a seamless, interactive experience for the user, from providing a GitHub link to having a fully installed project.

1. System Architecture and Workflow:

   The process begins with the user selecting their preferred LLM and providing a link to a GitHub repository. The system then initiates the following workflow:

   - Fetch README: The tool retrieves the README file content from the specified GitHub repository.
   - Initial Command Generation: The README content is passed to the selected LLM with a detailed prompt, requesting a sequence of installation commands. The LLM returns a list of initial commands.
   - Interactive Command Execution: The user is presented with the generated commands and can choose to execute, skip, edit, or quit for each command.
   - Error Feedback Loop: If a command results in an error, the error message (stderr) is captured and sent back to the LLM along with command history restored simultaneously. The LLM then attempts to generate a new, corrected command. Users can also add a prompt to help generation.

- Completion: The process ends when all installation steps are successfully executed, indicated by the LLM returning a "DONE_SETUP_COMMANDS" message.

2. Prompt Engineering:

   A key component of this project is the sophisticated prompt engineering used to guide the LLM in generating accurate and executable commands. The prompts are dynamically generated based on the user's system information (OS, architecture, Python version) and the specified installation directory.

   The initial prompt instructs the LLM to act as a "professional development environment configuration assistant" and provides a set of strict rules to follow:

   - Generate a command sequence for the user's specific operating system.
   - Recommend the use of conda for creating virtual environments. Handle placeholders for user-specific information (e.g., API keys).
   - Acknowledge that each command runs in a new terminal, requiring the chaining of commands (e.g., cd and pip install) using &&.
   - For Windows systems, ensure that changing directories is handled correctly.
   - Break down complex setups into individual, executable commands.

   Similarly, the "continue prompt" is engineered to handle errors and subsequent steps by feeding the output of the last command back to the LLM, enabling it to self-correct.

3. Implementation Details:

   - Backend: The tool is built using Python 3.7+ and managed with a requirements.txt file for dependencies.
   - API Integration: It supports multiple LLM providers, including Alibaba's DashScope (Qwen models) and Google's Gemini API, with API keys managed through a .env file.
   - Command Execution: The system executes commands and captures their stdout and stderr, which is crucial for the interactive feedback loop.
   - Project Structure: The codebase is organized into modules for configuration, GitHub utilities, LLM provider interactions, and command execution, making it modular and maintainable.

3 Challenges

- Command Execution Environment: A primary challenge was the stateless nature of standard Python subprocess functions like subprocess(), which execute each command in a new, isolated shell process. This meant that environment changes, such as directory navigation (cd) or conda environment activation, were not persistent across commands. To overcome this, the system had to be designed to re-establish the correct context (file path, virtual environment) for every single command, often by chaining commands together with &&.

- Version Mismatches: For older GitHub projects, the team faced issues with dependency version conflicts. As libraries and packages are continuously updated, the installation instructions in old README files could lead to errors that required manual intervention or more sophisticated error handling.

- User Expertise: To ensure a smooth installation, especially when errors occur, the user needs a basic understanding of how to prompt the LLM to correct those errors effectively. We can expose the plants to the user and provide prompt template, For easier and efficient input.

- Cross-Language Compatibility: While the tool demonstrates high efficacy for Python-based projects, its performance is currently limited when handling languages like C++ that involve more complex, platform-dependent build systems. A key priority for future development is to engineer a more language-agnostic framework, enabling robust support for a diverse range of programming environments.

- Evalution Criteria: Evaluating the performance of our LLM-driven installer requires nuanced metrics beyond a simple pass/fail. For this study, we used two primary criteria: Overall Completion Rate and First-Attempt Success Rate.

  - Overall Completion Rate is a binary measure (success/failure) indicating whether the project was fully installed by the end of the process, including any necessary error-correction loops.
  - First-Attempt Success Rate measures the initial accuracy of the LLM, calculated as the percentage of commands that executed successfully on the first try without intervention.

  Our initial tests were conducted on a curated set of popular GitHub repositories. We acknowledge that the total number of commands can vary per run due to the LLM's dynamic nature. For a more rigorous and standardized assessment in the future, we need to benchmark our tool against established frameworks like SWE-bench.

4 Results, and discussions

The project's effectiveness was evaluated by testing it on a variety of GitHub repositories choosing by hand, including those for computer vision (CV) and natural language processing (NLP), also on different system (both Windows and Linux environments.)

The evaluation results show a promising success rate, the outcome can be divided into three cases:

- Successful Installations: The tool successfully installed several projects, such as vggt, bert, and glances, with minimal to no errors. For instance, the vggt project was installed on Windows with a perfect 4/4 commands executed correctly.

- Handled Errors: In the case of installing bert, the tool encountered a TensorFlow version error, but the LLM was able to generate corrective commands, leading to a successful

installation with 14 out of 16 commands being correct in the end. For the glances project, it correctly identified a source not found error on Linux and fixed it by using a bash -c command.

- Unresolved Errors: Some complex projects with C++ dependencies or specific Torch version requirements proved more challenging. For example, the installation of mast3r failed at the wheel-building stage for a C++ dependency, and another project failed due to a Torch version error.

More details can be found in the following table.

Model	**Types**	**Completion**	**Error**	**Correct/Total**
vggt	cv/windows	yes	no error	4/4
mast3r	cv/windows/C++	no	stop at: Building wheel for asmk (`setup.py`): finished with status 'error'	9/?
bert	nlp/windows	yes	tensorflow version error; scikit has been deprecated	14/16
segmentation	cv/windows	no	torch version error	
mnist	cv/windows	yes	no error	6/6
glances	venv/Linux	yes	source not found, use `bash -c` command	5/6
ai-hedge-fund	LLM/Linux	yes	python3.11 not installed;	8/8
free-python-games	python/windows	yes	no error	5/5

Table 1: Summary of GitHub Model Installation Results

These results indicate that the LLM-based approach is highly effective for a wide range of standard Python projects. The interactive error-correction loop is a powerful feature that can overcome common setup issues. However, the system often fails to install projects correctly when dealing with complex non-Python dependencies (such as those in C++) or highly specific version constraints.Which can be improved in future.

5 Contribution Division

1. coding:
   - prompt: Xun Zhang, Zixuan Wang
   - user interface: Zixuan Wang,Haotian Shen
   - function implementation: Haotian shen, Xun Zhang
2. evaluation: Xun Zhang, Haotian Shen
3. presentation: Zixuan Wang