

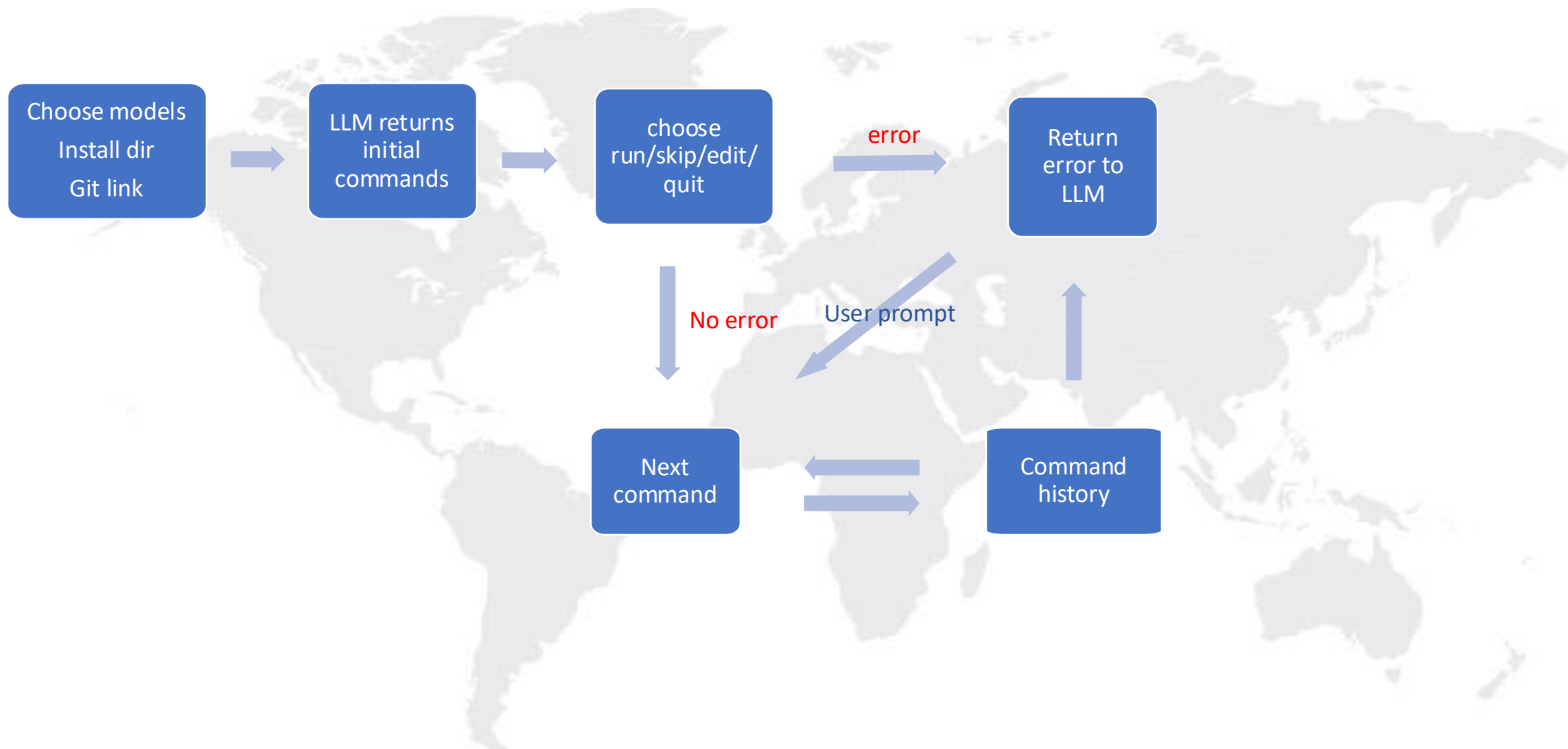
# LLM-Github-Installer

June 2, 2025

Haotian Shen, Xun Zhang, Zixuan Wang



# Workflow

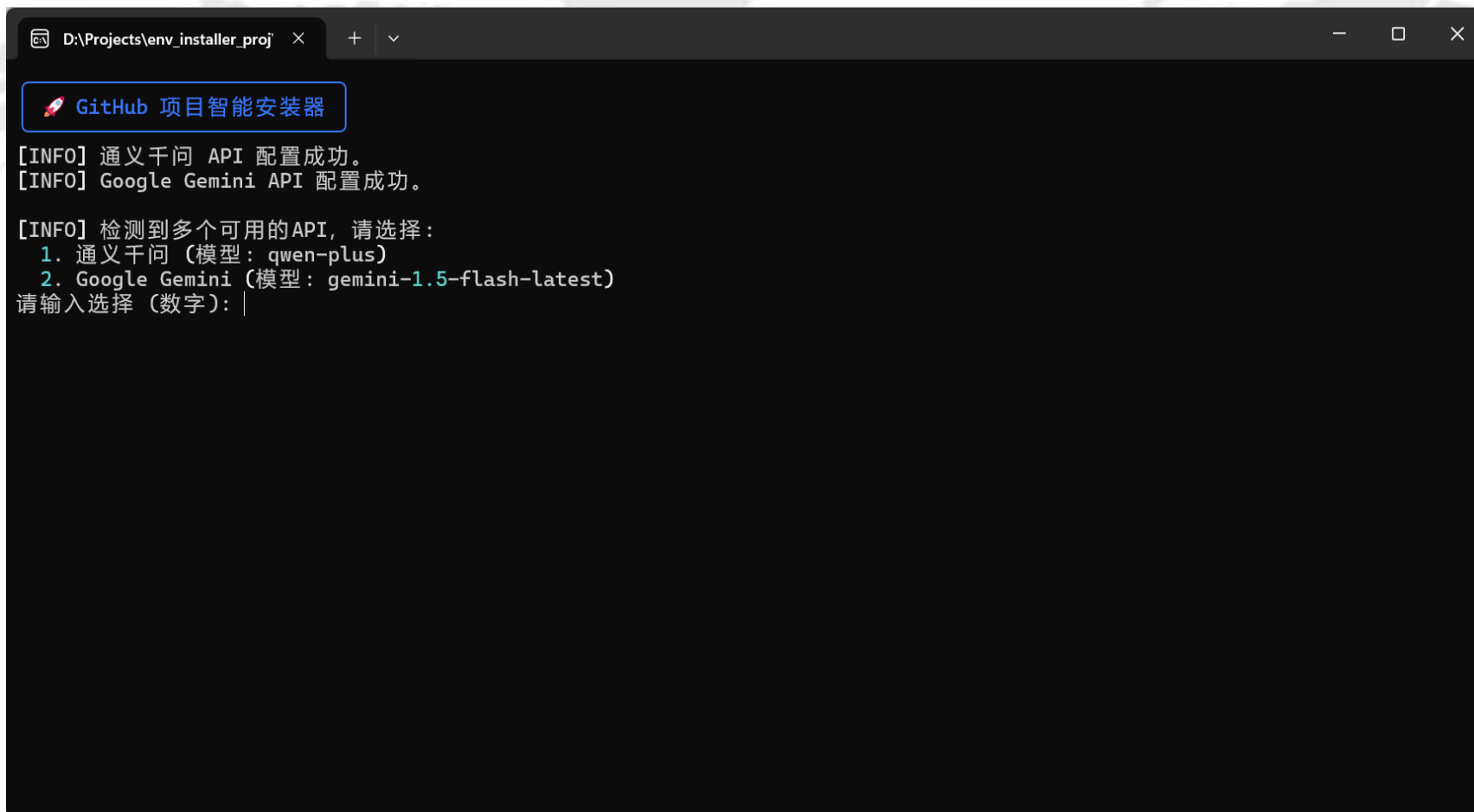


# Major Challenges

1. We don't have a proper command function in python to implement environment installation. For example, one common function *subprocess()*, they will call a brand new command line whenever they are called. This means we will lose our file location, conda activation, and other setup. So every time we need to input all the information we need to make sure all the commands are called in the correct file path, and all the libs are installed in the correct virtual environment.
2. There's some version mismatches for some early projects, since pip libraries are updating. Some
3. To ensure a fluent installation, users need have a basic knowledge to prompt LLM to correct errors.

# Implementations

Our projects are integrated into an executable file, making it just like using command.  
This is more streamlined and user-friendly.



```
D:\Projects\env_installer_proj x + v

GitHub 项目智能安装器

[INFO] 通义千问 API 配置成功。
[INFO] Google Gemini API 配置成功。

[INFO] 检测到多个可用的API, 请选择:
1. 通义千问 (模型: qwen-plus)
2. Google Gemini (模型: gemini-1.5-flash-latest)
请输入选择 (数字): |
```

# Implementation details --prompt

```
def _get_initial_prompt(self, readme_content: str) -> str:
```

```
    """获取初始安装命令的提示词"""
```

```
    return f"""你是一个专业的开发环境配置助手。请根据GitHub项目的README文件，为用户生成详细的安装和配置命令序列。
```

当前系统信息：

- 操作系统：{self.system\_info['os']}
- 架构：{self.system\_info['architecture']}
- Python版本：{self.system\_info['python\_version']}
- 用户指定的安装目录：{self.install\_directory}

请遵循以下规则：

1. 你需要生成一系列命令来安装和配置项目，确保每个命令都能在用户的系统上正确执行。
2. 如果项目有requirements.txt，使用pip安装依赖，推荐使用conda或者uv创建虚拟环境。如果没有说要安装python环境，则不需要用conda
3. 如果项目需要特殊配置，请明确指出
4. 命令应该适用于{self.system\_info['os']}系统，如果是Linux系统，请将shell变成bash，或者使用bash -c命令来完成
5. 每行只包含一个命令
6. 如果需要用户提供信息（如API密钥等），使用<YOUR\_XXX\_HERE>格式占位符
7. 如果设置完成，最后一行返回 "DONE\_SETUP\_COMMANDS"
8. \*\*\*重要：所有操作都应该在用户指定的安装目录 {self.install\_directory} 中进行
9. 重要：你的每一行都会新开一个terminal，这会导致原本这一行的命令出现问题，所以请你把原本命令所需要的一些前置命令都输出（就像第9条规则，或者cd命令进入安装目录然后），并且用&&进行连接，
10. 前一个规则基础上\*\*\*请检查所有生成的命令，每一个命令都需要重新进入项目所在的文件夹，然后，每当生成pip install 或者 conda install 命令时，请先激活环境，并用&&把所有的命令连接起来，例如：cd {self.install\_directory} && conda activate myenv && pip install -r requirements.txt
11. 在前一个规则基础上\*\*\*如果需要克隆项目，请克隆到指定的安装目录 {self.install\_directory} 中，另外你在git clone的时候需要先用cd命令进入这个文件夹，然后再通过&&把git clone在后面串联起来（比如cd install\_directory && git clone URL），或者对于pip install -r requirements.txt，你需要先cd到这个文件夹下，然后在cd命令后加上&& pip install ...
12. 尽量将命令拆分开来（如果要用&&连接则不用拆分）
13. windows系统下，所有cd 命令之前，都需要再加上一个目录名字，例如你想进入d盘，请使用cd d: && d:，请注意是所有命令，包括类似 cd d: && d: && conda activate myenv && pip install 这样的命令都需要在前面加上cd d: && d:，请注意是所有命令，包括类似 cd d: && d: && conda activate myenv && pip install -r requirements.txt 这样的命令都需要在前面加上cd d: && d:。
14. 如果需要用户输入一些api、路径之类的自定义的内容，请用 <YOUR\_VALUE\_HERE>，便于识别和重新生成！
15. \*\*\*如果项目需要多个API密钥或配置项，请将每个API的设置分成独立的命令。例如：

- 错误示例：export API\_KEY1=<YOUR\_API\_KEY1\_HERE> && export API\_KEY2=<YOUR\_API\_KEY2\_HERE>

- 正确示例：

```
export API_KEY1=<YOUR_API_KEY1_HERE>
```

```
export API_KEY2=<YOUR_API_KEY2_HERE>
```

如果要把API存储，请你写下进一步的命令

仔细阅读下面项目README内容，提取出重要安装信息：

```
{readme_content}
```

请分析该项目并生成安装配置命令序列，直接返回命令列表，每行一个命令，不要添加额外的解释文本："""

# Implementation details --prompt

```
def _get_continue_prompt(self, last_command: str, stdout: str, stderr: str, prompt_form_user: str) -> str:
    """获取继续执行的提示词"""
    base_prompt = f"""上一个命令: {last_command}
    执行结果:
    stdout: {stdout}
    stderr: {stderr}

    请基于执行结果决定下一步操作:
    1. 如果执行成功且还需要更多步骤, 请提供下一批命令
    2. 如果执行失败, 请提供修复命令, 如果返回的错误是找不到文件, 请注意, 每一次运行命令时, 都会相当于新建一个终端, 因此需要该命令的所有前置命令, 比如需要重新进入项目所在的文件夹,
    而且每当生成pip install 或者 conda install 命令时, 请先激活环境。所以, 在原来的命令上, 用&&把所有的命令连接起来,
    例如: cd {self.install_directory} && conda activate myenv && pip install -r requirements.txt
    3. 如果所有步骤都已完成, 请返回 "DONE_SETUP_COMMANDS"
    4. ***重要: 记住用户指定的安装目录是 {self.install_directory}, 所有必须要在该目录下运行的命令都需要在前面加上(cd install_directory && command...)
    请直接返回命令列表, 每行一个命令, 不要添加额外的解释文本: """

    if prompt_form_user:
        return base_prompt + f"\n同时请注意: {prompt_form_user}"
    return base_prompt
```

# Implementation details

```
owner, repo_name, readme = get_github_readme_content(github_project_url)

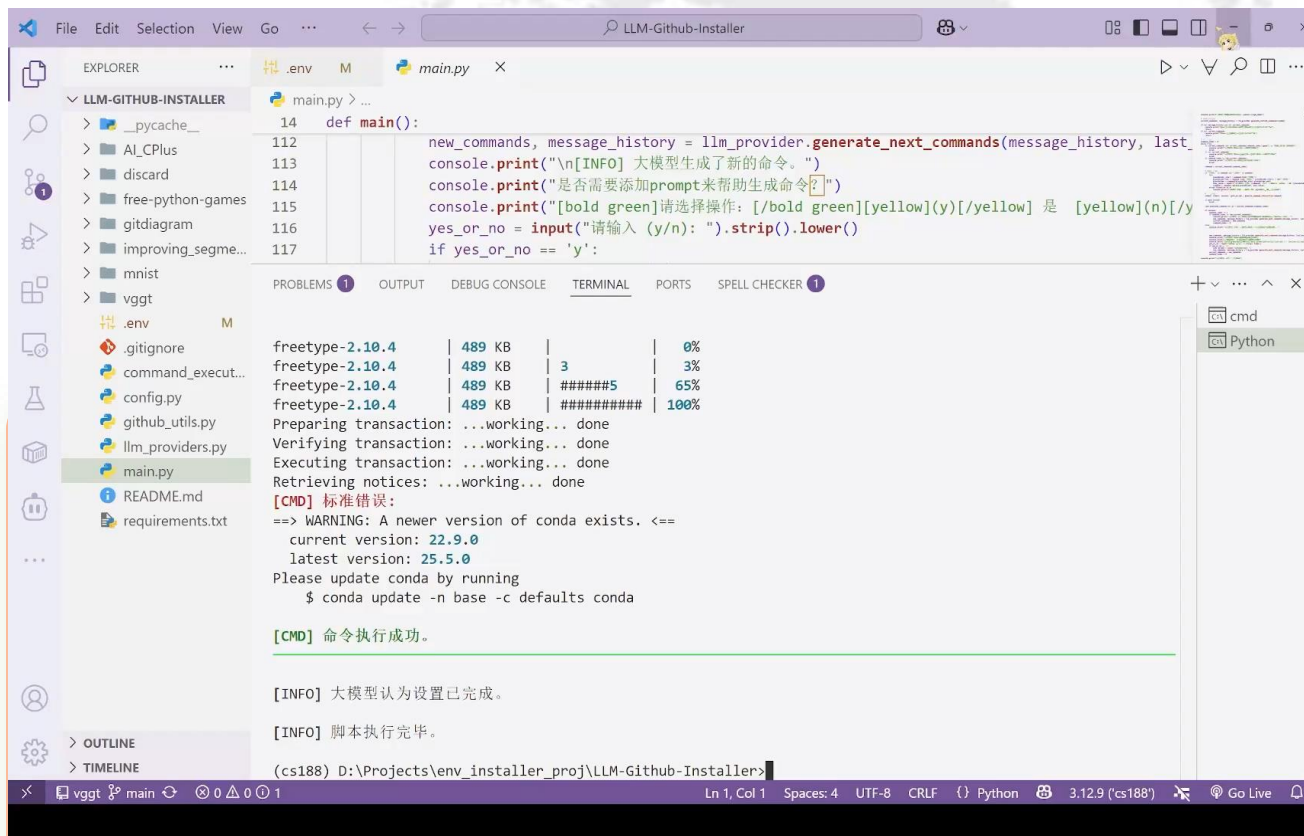
current_commands, message_history = llm.generate_initial_commands(readme, owner, repo_name)

While there is command left before "DONE_SETUP_COMMANDS":
    command = current_commands[command_index]
    stdout, stderr, success= execute_command (command)
    if last_command success:

        command_index += 1
        #check if we can finish
        if command_index ≥ len(current_commands):
            # here llm will check whether the set is done.We use prompt to tell model to generate "DONE_SETUP_COMMANDS"
            new_commands, message_history = llm.generate_next_commands(message_history, last_command, stdout, stderr)
            current_commands=new_commands
            command_index=0
        else:
            new_commands, message_history = llm.generate_next_commands(message_history, last_command, stdout, stderr)
            if user wants to add a prompt:
                new_commands, message_history = llm.generate_next_commands(message_history, last_command, stdout, stderr, user_prompt)
            current_commands=new_commands
            command_index=0
```



## Installing BERT on Windows system Demo



```

def main():
    new_commands, message_history = llm_provider.generate_next_commands(message_history, last_
    console.print("\n[INFO] 大模型生成了新的命令。")
    console.print("是否需要添加prompt来帮助生成命令?")
    console.print("[bold green]请选择操作: [/bold green][yellow](y)/[yellow] 是 [yellow](n)/[y
    yes_or_no = input("请输入 (y/n): ").strip().lower()
    if yes_or_no == 'y':

```

```

freetype-2.10.4 | 489 KB | 0%
freetype-2.10.4 | 489 KB | 3%
freetype-2.10.4 | 489 KB | #####5 | 65%
freetype-2.10.4 | 489 KB | ##### | 100%
Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
Retrieving notices: ...working... done
[CMD] 标准错误:
==> WARNING: A newer version of conda exists. <==
current version: 22.9.0
latest version: 25.5.0
Please update conda by running
$ conda update -n base -c defaults conda

[CMD] 命令执行成功。

[INFO] 大模型认为设置已完成。











[INFO] 脚本执行完毕。

(cs188) D:\Projects\env_installer_proj\LLM-Github-Installer>

```

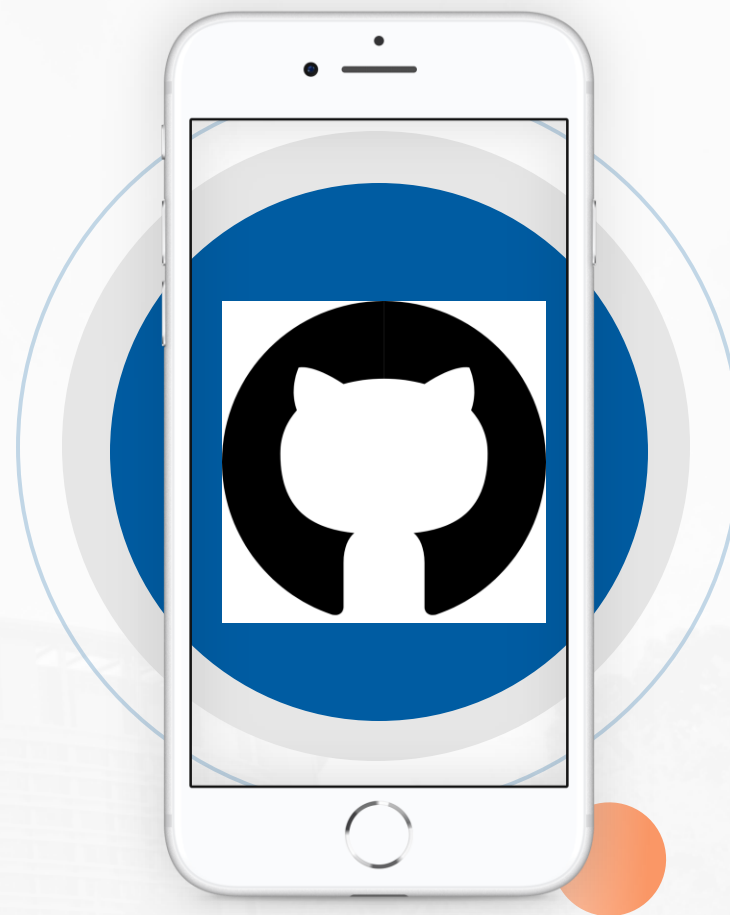


# Evaluation

Model	Types	Completion	Error Type	correct commands/total commands
 GitHub <b>GitHub - facebookresearch/vggt: [CVPR 2025 Best Paper Award ...</b>	cv/windows	yes	no error	4/4
 GitHub <b>GitHub - naver/mast3r: Grounding Image Matching in 3D with M...</b>	cv/windows/C++	no	stop at: Building wheel for asmk (setup.py): finished with status 'error'	9/?
 GitHub <b>GitHub - google-research/bert: TensorFlow code and pre-train...</b>	nlp/windows	yes	tensorflow version error; scikit has been deprecated	14/16
 GitHub <b>GitHub - lhoyer/improving_segmentation_with_selfsupervised_d...</b>	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 to	5/6
 ai-hedge-fund 		yes	python3.11 not installed Can request API !	8/8

# Future improvement

1. Better prompt engineering to enhance the installation success rate.
2. Making it a web version, so it's more and user-friendly.
3. Pay more attention to user's prompts.
4. Making different templates to support different languages.
5. Support more LLM.





**Thanks**