CSR-Bench: Benchmarking LLM Agents in Deployment of Computer Science Research Repositories

Yijia Xiao[†*] **Runhui Wang**[‡*] **Luyang Kong**[‡*] **Davor Golac**[‡] **Wei Wang**[†]
[†]University of California, Los Angeles
[‡] Amazon Web Services
* Equal Contribution
{yijia.xiao,weiwang}@cs.ucla.edu
{runhuiw,luyankon,dgolac}@amazon.com

Abstract

The increasing complexity of computer science research projects demands more effective tools for deploying code repositories. Large Language Models (LLMs), such as Anthropic Claude and Meta Llama, have demonstrated significant advancements across various fields of computer science research, including the automation of diverse software engineering tasks. To evaluate the effectiveness of LLMs in handling complex code development tasks of research projects, particularly for NLP/CV/AI/ML/DM topics, we introduce CSR-Bench, a benchmark for Computer Science Research projects. This benchmark assesses LLMs from various aspects including accuracy, efficiency, and deployment script quality, aiming to explore their potential in conducting computer science research autonomously. We also introduce a novel framework, CSR-Agents, that utilizes multiple LLM agents to automate the deployment of GitHub code repositories of computer science research projects. Specifically, by checking instructions from markdown files and interpreting repository structures, the model generates and iteratively improves bash commands that set up the experimental environments and deploy the code to conduct research tasks. Preliminary results from CSR-Bench indicate that LLM agents can significantly enhance the workflow of repository deployment, thereby boosting developer productivity and improving the management of developmental workflows.

1 Introduction

With the rapid evolution of Large Language Models (LLMs), it has been demonstrated that LLMs have increased reasoning ability over the last few years, making intelligent agents based on LLM possible. Current agent-related applications in computer science include code writing (Codex (Chen et al., 2021), Deepseek Code (Guo et al., 2024), CodeLLaMA (Rozière et al., 2023), etc.), code base generation (MetaGPT (Hong et al., 2023), Agentless (Xia et al., 2024a) and CodeStar(Li et al., 2023)), code correction (SWEBench (Jimenez et al., 2024)), and more.

In computer science research projects, the associated codebases grow very rapidly, and a self-consistent codebase typically has several parts, including the instruction file (e.g., the README file), associated code packages, and related data. The instruction file usually contains an overview of the project, including environment setup, data preparation (e.g., data download and pre-processing, model weights download and preparation), model training process, performance evaluation, and the setup of a demo project (e.g., a chatbot project that interacts with users). In computer science, prestigious conferences, including NAACL, ACL, ICLR, NeurIPS, CVPR, KDD, etc., encourage researchers to release source code for reproducibility of their accepted papers, and GitHub is the top choice for maintaining codebases for most researchers. A major step of computer science research is reproducing existing work, which is essential to gain insights and propose novel methodologies. However, even for well-documented and self-consistent projects, the setup process requires manual efforts and cannot be fully automated; many steps of setting up a code repository are rather mechanical, such as installing/updating dependency packages to configure the environment, downloading data, updating the relevant script/data directories, etc., which is tedious and often time-consuming.

To tackle such challenges, we propose to use LLM agents for automating the deployment of code repositories of research projects, and build a benchmark, **C**omputer **S**cience **R**esearch **Bench**mark (CSR-Bench) for evaluating LLM Agents on code repository deployment tasks. We also propose a multi-agent collaborative framework, CSR-Agents, to automate the deployment of code repositories by coordinating multiple LLM agents with different

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 12705–12723

expertise and iterative improvement with provided tools. From more than ten major NLP/CV/AI/ML/DM conferences, we collected the 100 highly rated repositories[1], which are carefully selected from an initial candidate set that has more than 1500 top-star repositories. Our selection criteria include topic diversity and self-containment[2] so that CSR-Bench can provide a comprehensive evaluation of LLM agents on code deployment tasks including instruction generation, command execution, and self-improvement with tools.

To the best of our knowledge, CSR-Bench is the first benchmark for the deployment of computer science research projects, providing a reference for evaluating LLM agents. We note that the success of code deployment depends not only on code generation, but also on many non-coding factors including tasks like experimental environment setup, data/model preparation, correcting bash commands, searching for solutions, and etc. Our proposed multi-agent framework, CSR-Agents, aims to achieve automation of code deployment, which can accelerate the progress for computer science research projects.

Our contributions are as follows:

- We introduce CSR-Bench to assess LLM's ability to understand instruction manuals and complex project structures, generate executable commands for code deployment, and solve errors during deployment.

- We propose the CSR-Agents framework, which leverages multi-agent cooperation with specialized capabilities including instruction comprehension, command execution, error log analysis, and error correction with searching and retrieval tools.

- We design a standardized testing system for reproducibility in CSR-Bench, which can make CSR-Agents a CI/CD[3] standard system of computer science code repository deployment, ensuring ease of use, reusability, and improving communication and collaboration efficiency in computer science research projects.

- We evaluated a wide range of foundation models for CSR-Agents on CSR-Bench. Results indicate that LLM agents can potentially accelerate the process of repository deployment, thereby boosting researcher productivity. However, it is still challenging to achieve full automation.

[1]With appropriately permissive licenses.

[2]The information within the repository is mostly sufficient for a successful deployment.

[3]Continuous Integration and Continuous Delivery.

2 Related Work

Coding LLMs. Large Language Models (LLMs) have become the go-to solution for a wide array of coding tasks due to their exceptional performance in both code generation and comprehension (Chen et al., 2021). These models have been successfully applied to various software engineering activities, including program synthesis (Patton et al., 2024; Chen et al., 2021; Li et al., 2022a; Iyer et al., 2018), code translation (Pan et al., 2024; Roziere et al., 2020, 2021), program repair (Xia et al., 2023; Xia and Zhang, 2023; Monperrus, 2018; Bouzenia et al., 2024), and test generation (Deng et al., 2023; Xia et al., 2024b; Deng et al., 2024; Lemieux et al., 2023; Kang et al., 2023). Beyond general-purpose LLMs, specialized models have been developed by further training on extensive datasets of open-source code snippets. Notable examples of these code-specific LLMs include CODEX (Chen et al., 2021), CodeLlama (Rozière et al., 2023), StarCoder (Li et al., 2023; Lozhkov et al., 2024), and DeepSeek-Coder (Guo et al., 2024). Additionally, instruction-following code models have emerged, refined through instruction-tuning techniques. These include models such as CodeLlama-Inst (Rozière et al., 2023), DeepSeek-Coder-Inst (Guo et al., 2024), WizardCoder (Luo et al., 2023), Magicoder (Wei et al., 2023), and OpenCodeInterpreter (Zheng et al., 2024).

Benchmarking LLM-based coding tasks. To assess the capabilities of LLMs in coding, a variety of benchmarks have been proposed. Among the most widely utilized are HUMANEVAL (Chen et al., 2021) and MBPP (Austin et al., 2021), which are handcrafted benchmarks for code generation that include test cases to validate the correctness of LLM outputs. Other benchmarks have been developed to offer more rigorous tests (Liu et al., 2023a), cover additional programming languages (Zheng et al., 2023; Cassano et al., 2023), and address different programming domains (Jain et al., 2024; Hendrycks et al., 2021; Li et al., 2022b; Lai et al., 2023; Yin et al., 2022).

More recently, research has shifted towards evaluating LLMs on real-world software engineering challenges by operating on entire code repositories rather than isolated coding problems (Jimenez et al., 2024; Zhang et al., 2023; Liu et al., 2023b). A notable benchmark in this area is SWE-bench (Jimenez et al., 2024), which includes tasks requiring repository modifications to resolve actual GitHub issues. The authors of SWE-bench have also released a more focused subset, SWE-bench Lite (swe, 2024), which contains 300 problems centered on bug fixing that only involves single-file modifications in the ground truth patches. ML-Bench (Liu et al., 2023c) is a benchmark for evaluating large language models and agents for Machine Learning tasks on reporitory-level code. It involves 18 repositories and focuses on code generation and interactions with Jupyter Notebooks.

Repository-level coding. The rise of agent-based frameworks (Xi et al., 2023) has spurred the development of agent-based approaches to software engineering tasks. Devin (dev, 2024) (and its open-source counterpart OpenDevin (ope, 2024b)) is among the first comprehensive LLM agent-based frameworks. Devin employs agents to first perform task planning based on user requirements, then allows them to use tools like file editors, terminals, and web search engines to iteratively execute the tasks. SWE-agent (Yang et al., 2024) introduces a custom agent-computer interface (ACI), enabling the LLM agent to interact with the repository environment through actions like reading and editing files or running bash commands. Another agent-based approach, AutoCodeRover (Zhang et al., 2024), equips the LLM agent with specific APIs (e.g., searching for methods within certain classes) to effectively identify the necessary modifications for issue resolution. Beside these examples, a variety of other agent-based approaches have been developed in both open-source (Gauthier, 2024) and commercial products (Bouzenia et al., 2024; Chen et al., 2024; Ma et al., 2024; lin, 2024; fac, 2024; ibm, 2024; ope, 2024a; mar, 2024; ama, 2024).

Unlike existing benchmarks and agent-based frameworks, which focus on the code generation/-completion tasks, our proposed CSR-Bench and CSR-Agents focus on the code deployment task, which is under-studied in the field.

3 CSR-Bench

In this section, we provide the problem statement for code deployment in CSR-Bench, introduce the code repository collection process of computer science research projects, and show their statistics.

3.1 Problem Statement

The CSR-Bench consists of a collection of computer science research repositories from GitHub and these repositories are used for evaluating the capabilities of LLMs in code deployment tasks. For each repository, the deployment tasks typically include: (1) setting up the environment; (2) preparing necessary data and model files; (3) conducting model training; (4) demonstration of inference; (5) performance evaluation. To complete these tasks, we prompt LLMs to generate executable bash commands by using the README file as the primary source of information and other repository contents (source code, bash scripts, directory structure, and etc.) as supplementary information.

Metric. During the evaluation in CSR-Bench, the large language model will be prompted to generate executable commands for the corresponding sections for each repository. we use the completion rate as a key metric, defined as the ratio between number of successfully executed commands and the total number of commands executed.

3.2 Repository Collection

In CSR-Bench, we aim to collect a diverse and comprehensive collection of code repositories of computer science-related research projects. GitHub is a good data source for this purpose and it provides tags for identifying most relevant repositories. Some example tags are “nlp”, “naacl”, and “emnlp2024”. Since CSR-Bench focuses on computer science-related repositories, we filter the repositories by tags of various conference names and categories to ensure they include diverse topics.

For repository selection, we use GitHub tags to obtain an initial set of over 1500 repositories that are relevant to computer science research topics and categorizing them into five areas: Natural Language Processing, Computer Vision, Large Language Models, Machine Learning, and Interdisciplinary topics. Notably, we collect repositories related to large language models because nowadays LLM-related research projects are increasingly popular due to its foundational impact in various areas of computer science.

We obtain 100 high-quality code repositories for CSR-Bench in the following steps. First, we categorize this initial set and sort them by the number of GitHub stars. Next, we manually check the content of each repository starting from the top of the sorted list. In this step, we only keep reposi-

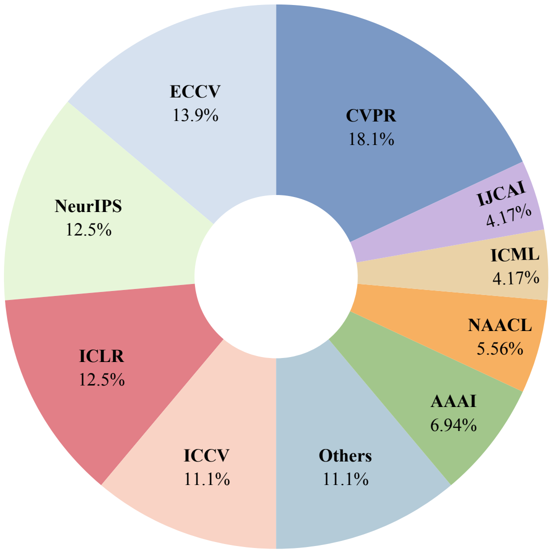

Figure 1: Conf Distribution of CSR-Bench

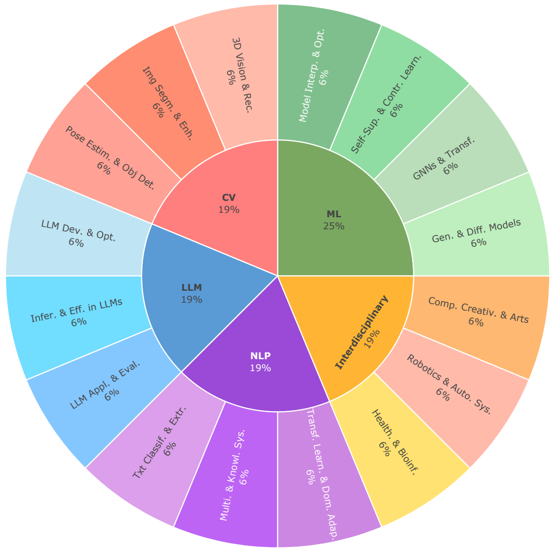

Figure 2: Topic Distribution of CSR-Bench

tories that contain sufficient information in their README files. We also skip the repositories that do not contain deployable code. Finally, we check the licenses of the repositories and make sure they are permissive.

3.3 Statistics of CSR-Bench

This section provides an in-depth analysis of the traits of repositories in CSR-Bench. We examine the diversity and breadth of topics covered, as well as detailed statistics about the documentation and structure of these repositories.

In CSR-Bench, the README files and directory structures provide critical insights into the usability and organization of repositories. We use the following figures to analyze the lengths of README file and number of files, and offer a quantitative view of content complexity and organizational depth. The length of README file is an important metric because the most LLMs have limits on the input token length. The number of files indicate the complexity of the code repository.

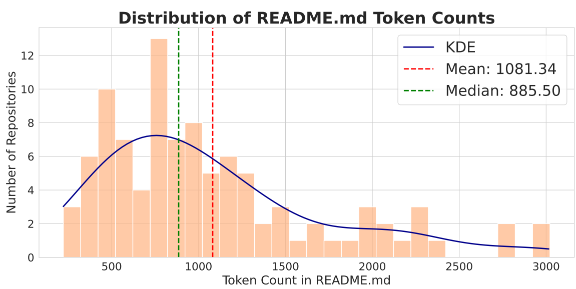

Figure 3: Number of Tokens per README

Figure 3 shows the distribution of token counts in README files, highlighting the extent of documentation, which is essential for user understanding and repository usability. Since the mean token counts is just over 1000 and the maximum counts is around 3000, most LLMs can take the full README files as input.

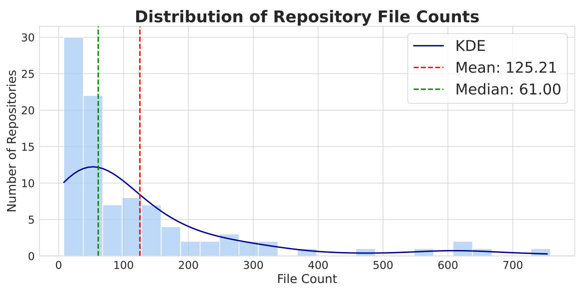

Figure 4: Number of Files per Repository

Figure 4 depicts the distribution of file counts in repositories, reflecting their complexity and scale based on the number of files. Because the number of files in most repositories are in the lower hundreds, it is feasible to leverage directory structure for code deployment with LLMs.

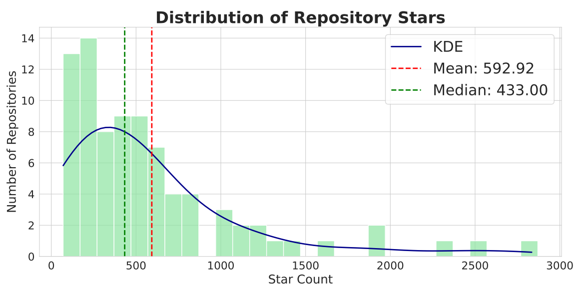

Figure 5: Stargazer Distribution

Figure 5 shows the distribution of star counts in selected repositories. The average count is over 590, which means that these repositories receives significant attention and indicates they are generally well maintained.

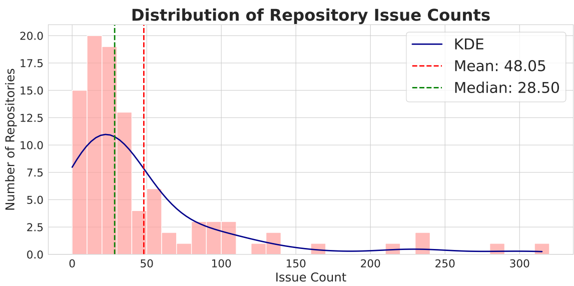

Figure 6: Size of Issue Database

Figure 6 shows the distribution of issues counts in selected repositories. On average, each repository contains over 48 issues, indicating these repositories have a good amount of engagements with the open-source community and sufficient support from the authors. Therefore, the information in the issues are valuable for the deployment tasks.

4 CSR-Agents

In this section, we propose CSR-Agents, a multi-agent framework that leverages LLMs for different tasks and achieves effective cooperation among agents for code deployment tasks. We introduce our standardized environment for code deployment, functionalities of different agents, and their cooperation workflow.

4.1 Code Deployment with LLM

For each repository, we use the README file under the root folder as the major source of information and prompt LLMs to generate executable bash commands for different steps of deployment including environment setup, data preparation, training, inference and evaluation.

To achieve reproducbility and safety, we use the Docker container to isolate the code deployment environment. We build a standard Docker image that is equipped with essential tools like bash, Conda, GCC, Make, Python, etc. for evaluations across all repositories and various LLMs. In the container, we use a counter to count the total number of scripts that needed to be executed and the number of scripts that were executed successfully. In this way, we safeguard the entire computing system, especially from bash command executions that involve system-level permissions and could potentially break the whole system.

We note that the use of Docker is not part of the original repositories. The introduction of Docker into our evaluation system can ensure that each time we conduct the evaluation, we start from the same environment for fair comparisons across different LLMs, and the evaluation of one repository does not affect any other repositories. Another benefit of using Docker is that we can speed up the evaluation by running different evaluations in separate docker instances at the same time.

The LLM generated bash commands may not work well for successful deployment due to various reasons. First, in some README files, some basic directives (such as `conda` and `pip`) were missing, so the LLMs could not generate these commands at the first attempt. Secondly, the installation of packages may not be completed in a single attempt and usually needs several iterations of trial-and-error. Thirdly, some steps of the deplopyment require checking additional information like GitHub issues of the corresponding repositories and from the internet. To handle these issues, we design a group of LLM agents that cooperate effectively.

4.2 CSR-Agent: LLM Agent Design

In CSR-Agents, we adopt an iterative trial-and-error process for successful code deployment. Specifically, LLM takes README, directory structure, and error logs from failed command execution as input and generates bash commands to complete the deployment tasks. The system comprises five agents: Command Drafter, Script Executor, Log Analyzer, Issue Retriver, and Web Searcher. These agents collectively facilitate the deployment of a code repository. The complete workflow is shown in Figure 7.

Command Drafter: This agent reads the README files and directory structure and generates a draft script containing bash commands for deployment. Then, it divides the entire script into five sections, each corresponding to a step in the code repository deployment. This sectional division also serves as an evaluation standard later on.

Script Executor: This agent receives the draft commands from the Command Drafter and execute the commands in our standardized Docker environment. After execution, it collects logs from `bash`, including standard output and errors. Note that during bash script execution, no explicit return code

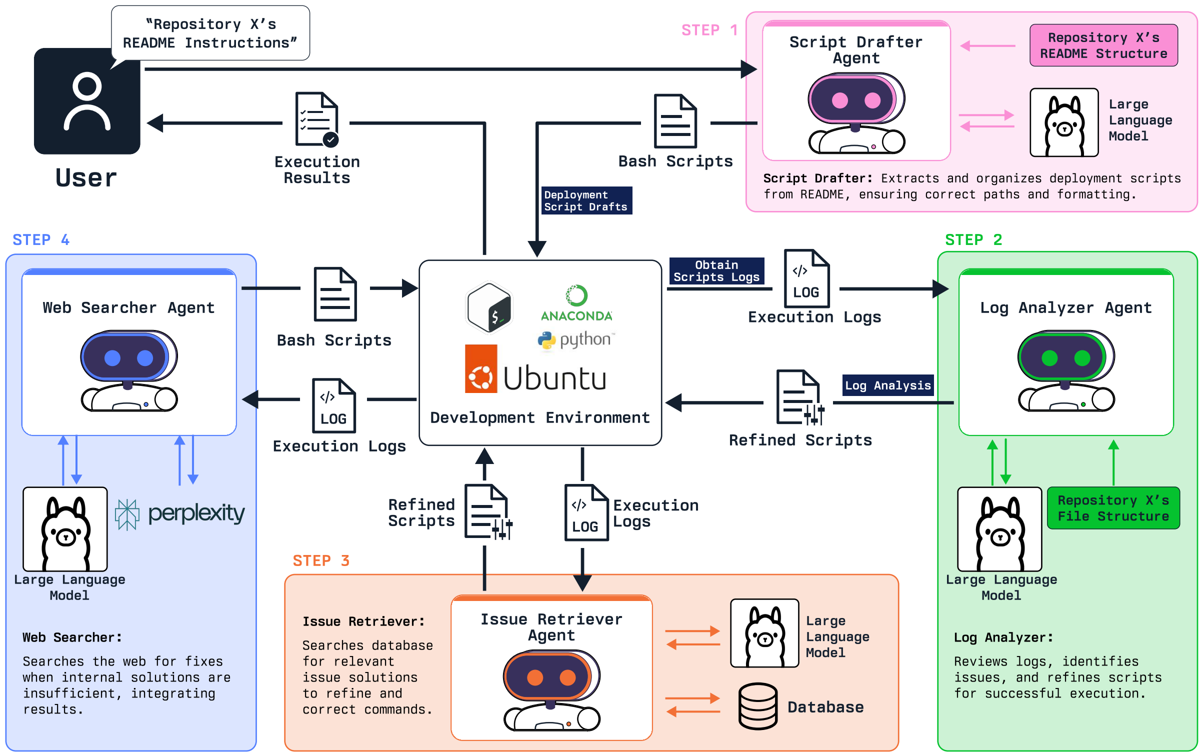

Figure 7: Workflow of CSR-Bench

is provided by bash. We experimented with setting predefined special prompt to bash and parse the return code from the returned output and error message. However, we find that a good quantity of commands do not have return code, making the feedback from bash not informative to other agents. To address this challenge, we leverage the LLM in the executor, instructing it to provide feedback based on the standard output and error messages. We then parse this feedback to generate a return code: if the return code is zero, then the command is executed successfully; otherwise, the command, output, and error messages are logged and sent to the Log Analyzer.

Log Analyzer: This agent reads the logs and the associated bash command, and checks for updates, missing prerequisites, or script paths that need updating. It also identifies any other missing components and returns a curated command for a successful execution. We note that this agent only reply on the internal knowledge of the LLM and may not be able to correctly solve the errors.

Issue Retriever: This agent takes in the command, standard output and error message and search them against the issue database we collected from the repository. It leverages RAG and its pipeline requires a search algorithm to query the input against the database. In our design, the queries are the combination of the commands executed, standard output and error messages. The database contains GitHub issues and their communication records for the repository. We experimented with BM25 and Contriever as the retrieval algorithm and decided to use BM25 for (1) BM25's higher search speed and (2) the fact that error logs and issues generally share keywords, so sentimental search do not possess much advantage over lexical search.

Web Searcher: This agent utilizes the Perplexity API to obtain solutions for failed execution. If the pipeline reaches this stage, it indicates that the Log Analyzer and Issue Retriever failed to solve a failed command. The standard output, standard error, and the failed command are fed to Perplexity to search the web for solutions. The Web Searcher then analyzes the solutions from Perplexity and generates new bash commands to resolve the issue.

4.3 LLM Coorporation Framwfork

The workflow operates as follows: Deployment commands are drafted from repository documentation, executed in a bash environment, and adjusted based on log analysis if errors arise. Additional information is retrieved from an issue database and web search if needed. The process concludes with a summary that outlines successes and identifies steps needing further attention.

Drafting the Initial Commands The instructions from each repository's README are fed into the Command Drafter LLM Agent to draft the necessary commands for deploying the repository. These commands are organized into five stages: prerequisite installation, data and model checkpoints downloading, training, inference, and evaluation. It is important to note that not every repository contains all these sections, and some sections may be empty.

Execution of Commands After the draft stage, the script drafted will be sent to the Script Executor. To provide LLM Agents with the Python interface, we implement a BashExecutor that encapsulates the bash binary executable file in it. If the commands execute successfully, the process is deemed successful and the deployment pipeline will return `True`. Though looks simple, the bash simulation was not straightforward during our experiments, since we need to consider the environment variables, stdout, stderr, etc. Initially, we used a subprocess to handle it, but the success rate was extremely low. Upon analyzing the logs, we discovered that no commands related to environment variable changes were reflected in subsequent instructions. For instance, if the Command changes the environment to a Python virtual environment or another `conda` environment, this change only applies to that specific command. Afterward, the Python interpreter and package manager revert to their default settings. Besides, not all command executions have a valid return code, therefore, we utilize an LLM to parse the standard output and standard error (if any) to obtain the return state of the execution.

Analyze the Execution Log If the execution is not successful and error occurs, the standard output, standard error, command, and return code are logged and analyzed. The Log Analyzer examines the error messages and attempts to refine the execution commands or adjust prerequisites to ensure the environment is prepared. We have a `max_attempt` argument that limits the number of retries for the log analyzer. If the issue persists after certain attempts, the workflow utilizes the web search tool to request external information.

Retrieve Augmented Generation from Issue Database After analyzing the execution log and making initial adjustments, the next step is to retrieve more insights using a Retrieve Augmented Generation (RAG) approach. The Issue Retriever agent uses the logged command, output, and error messages to query the issue database for similar past problems or discussions. Leveraging the BM25 retrieval algorithm, it matches keywords from the logs to relevant entries. If a match is found, the agent extracts solutions or troubleshooting steps, feeding them back into the workflow to refine commands. If no relevant match is found, the process escalates to the Web Searcher for external information.

Search the Internet for External Information Using tools such as Perplexity, the WebSearcher agent integrates the command error logs and standard output with external information. This process refines the command and retries execution in the bash with a limited number of attempts. The agent records the information and transfers it to the deployment summarizer if the issue remains unresolved.

5 Evaluations

In this section, we evaluate CSR-Agents in CSR-Bench with a wide range of popular foundation LLM families, including Claude [4], GPT-4 (Achiam et al., 2023), Llama-3 (Dubey et al., 2024), and Mistral [5]. For each LLM family, we experimented with different model sizes for thorough comparisons.

We show the completion rate in Table 1, Table 2, Table 3, and Table 4. In these tables, we use **S** to stand for the Setup stage, **D** to stand for the ownload stage, **T** to stand for the Training stage, **E** to stand for the Evaluation stage, and **I** for the inference stage.

5.1 Initial Drafter

As shown in Table 1, all models perform well on *Setup* and *Download* tasks (success rates around 0.23 to 0.28) but struggle with *Training*, *Evaluation*, and *Inference*, where success rates are close to zero. This indicates that the drafter agent handles basic installation effectively but has difficulty with complex tasks requiring updates to file paths and environment variables.

We note that this is also similar to the deployment process of a real researcher, where their first execution is more likely to fail and they need to analyze the errors and leverage tools like GitHub issues or search engines to solve the problem.

[4] https://www.anthropic.com/news/claude-3-family
[5] https://mistral.ai/technology/#models

Table 1: Drafter Success Metrics of Different Models.

Model Type	Name	S	D	T	E	I
Claude	Instant	0.232	0.189	0.007	0.000	0.000
	3-Haiku	0.253	0.239	0.046	0.005	0.052
	3-Sonnet	0.284	0.283	0.045	0.024	0.031
GPT	4o-Mini	0.242	0.229	0.008	0.016	0.029
	4o	0.261	0.238	0.039	0.022	0.031
	4-Turbo	0.271	0.252	0.028	0.039	0.032
Llama	3-70B	0.239	0.306	0.019	0.040	0.032
	3.1-8B	0.243	0.200	0.051	0.037	0.007
	3.1-70B	0.260	0.280	0.032	0.022	0.019
Mistral	Large	0.243	0.266	0.047	0.031	0.026
	Large-2	0.251	0.279	0.039	0.025	0.024

5.2 Log Analyzer

Table 2 shows noticeable improvements in all tasks compared to the drafter stage. Success rates for *Setup* and *Download* increase to around 0.34 to 0.40, while complex tasks see gains up to 0.18.

Table 2: Analyzer Success Metrics of Different Models.

Model Type	Name	S	D	T	E	I
Claude	Instant	0.342	0.353	0.104	0.109	0.151
	3-Haiku	0.350	0.301	0.132	0.037	0.130
	3-Sonnet	0.388	0.400	0.168	0.116	0.129
GPT	4o-Mini	0.347	0.317	0.118	0.078	0.131
	4o	0.362	0.353	0.148	0.115	0.145
	4-Turbo	0.353	0.322	0.161	0.094	0.148
Llama	3-70B	0.361	0.382	0.111	0.185	0.176
	3.1-8B	0.304	0.386	0.183	0.114	0.123
	3.1-70B	0.313	0.335	0.141	0.182	0.151
Mistral	Large	0.324	0.349	0.121	0.143	0.144
	Large-2	0.340	0.357	0.152	0.199	0.163

Analyzers leverage dynamic feedback from the execution of commands to refine scripts and try to correct errors from executed commands.

5.3 Issue Retriever

In the Issue Retriever stage (Table 3), success rates continue to improve, especially for complex tasks like **Training**, **Evaluation**, and **Inference**, reaching up to 0.25.

The results show that access to a knowledge base with informative discussions on issues of the code repository allows LLMs to retrieve solutions to execution errors from earlier stages, enhancing performance in complex operations.

5.4 Web Searcher

The Searcher Success Metrics in Table 4 exhibits the highest performance. Success rates for *Setup* and *Download* reach up to 0.46, and complex tasks improve to between 0.15 and 0.29.

Table 3: Issue Retriever Success Metrics of Different Models.

Model Type	Name	S	D	T	E	I
Claude	Instant	0.365	0.369	0.129	0.130	0.169
	3-Haiku	0.374	0.329	0.139	0.061	0.143
	3-Sonnet	0.442	0.436	0.254	0.183	0.163
GPT	4o-Mini	0.375	0.367	0.171	0.122	0.159
	4o	0.379	0.375	0.169	0.128	0.160
	4-Turbo	0.377	0.381	0.178	0.126	0.164
Llama	3-70B	0.364	0.399	0.113	0.159	0.154
	3.1-8B	0.305	0.389	0.182	0.115	0.122
	3.1-70B	0.312	0.334	0.143	0.179	0.153
Mistral	Large	0.357	0.358	0.174	0.155	0.153
	Large-2	0.359	0.380	0.181	0.155	0.152

Table 4: Searcher Success Metrics of Different Models.

Model Type	Name	S	D	T	E	I
Claude	Instant	0.388	0.406	0.151	0.131	0.190
	3-Haiku	0.385	0.338	0.155	0.070	0.163
	3-Sonnet	0.467	0.467	0.291	0.194	0.189
GPT	4o-Mini	0.412	0.405	0.201	0.131	0.179
	4o	0.415	0.407	0.198	0.130	0.183
	4-Turbo	0.416	0.406	0.200	0.133	0.182
Llama	3-70B	0.380	0.442	0.200	0.173	0.170
	3.1-8B	0.318	0.447	0.201	0.157	0.165
	3.1-70B	0.344	0.463	0.183	0.196	0.184
Mistral	Mistral-Large	0.375	0.450	0.199	0.174	0.168
	Mistral-Large-2	0.373	0.452	0.201	0.172	0.169

Web search enables models to find up-to-date solutions, resolving issues that previous agents could not, leading to substantial improvements.

5.5 Aggregated Results

We show the aggregated results of a single LLM on different tasks with different level of engagements of multi-agents in Figure 8, Figure 9, Figure 10, and Figure 11. In short, with more agents contributing to solving the tasks, the success rate increases across all tasks for all LLMs, which demonstrates the effectiveness of our proposed CSR-Agents.

5.6 Results Interpretation

The evaluation of the CSR-Bench involved assessing the performance of various large language models (LLMs) across key tasks: Setup, Download, Training, Inference, and Evaluation. These tasks are essential for deploying repositories within the CSR-Bench.

5.6.1 Task-Specific Outcomes

Setup and Download: Most models consistently performed well, reflecting their capability to initiate and manage basic deployment processes.

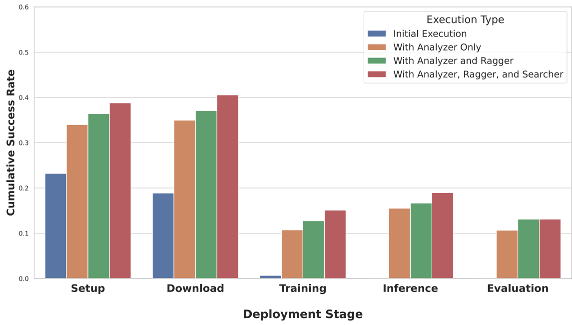

Figure 8: Performance of Claude 3 Sonnet

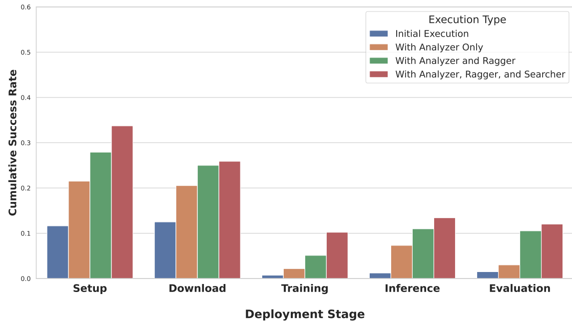

Figure 9: Performance of GPT 4o

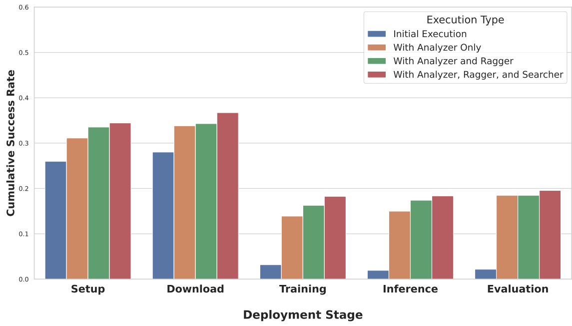

Figure 10: Performance of Llama 3.1 70B Instruct

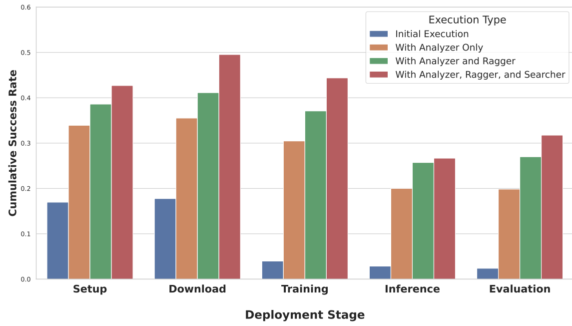

Figure 11: Performance of Mistral Large 2

Inference and Evaluation: Performance was less consistent, with some models demonstrating moderate success, but generally struggling with the complexity of these tasks.

Training: Training tasks are particularly challenging, with lower success rates across the board, indicating that current LLMs require further refinement to handle training processes effectively.

5.6.2 Overall Performance

The success metrics across different tasks and models indicate a wide variability in performance. Generally, models showed higher success rates in Setup and Download tasks, with performance tapering off in more complex tasks such as Inference, Evaluation, and Training. This pattern highlights the challenges LLMs face in handling the full deployment process autonomously.

The results demonstrate that while LLMs have made significant strides in automating repository deployment, their ability to manage complex tasks remains limited. Improvements are needed, particularly in the areas of Inference and Training, to achieve fully autonomous and reliable deployment of science repositories.

However, there is still a large gap between LLMs and real scientists even if the advnanced tools are provided to the LLMs. To explain, it is not trivial to handle the nuances in the experiement environment setup for the science repositories. For example, the hardware and software compatibility issues are very common in code deployment and often causes confusions even for domain experts.

6 Conclusion

The work introduces CSR-Bench, a benchmark designed to evaluate the capabilities of LLM agents in automating the deployment of GitHub repositories for scientific research. Our study highlights that while LLMs show potential in handling tasks like environment setup and data preparation, they face challenges in complex tasks such as training and inference, where success rates are notably lower.

Our multi-agent framework, CSR-Agents, exemplifies how LLMs can collaborate to tackle deployment challenges, offering a promising approach to improving automation in software engineering. However, the results indicate that further advancements are needed to fully realize autonomous and reliable deployment processes.

Overall, CSR-Bench serves as a crucial tool for assessing and improving LLM-driven deployment workflows in scientific research, paving the way for more efficient and automated computer science projects exploration.

Limitations

Although our benchmark framework supports several tools to facilitate large language model agents in the code deployment task, it does not actually improve the original reasoning capabilities of the large language models that are used in the agents. To improve LLMs' reasoning capabilities for this specific task, the community may resort to techniques like RLHF, which is orthogonal to this work. Our benchmark only focuses on code repositories that related to computer science research topics, and does not involve other types of repositories. Although this framework can be reused for other types of the repositories, we do not explore that direction in this work, and leave it to future works.

References

2024. Agent-101: A software engineering agent for code assistance developed by ibm research. https://github.com/swe-bench/experiments/blob/main/evaluation/lite/20240612_IBM_Research_Agent101/README.md/.

2024. Amazon q developer the most capable generative ai–powered assistant for software development. https://aws.amazon.com/q/developer/.

2024. Devin, ai software engineer. https://www.cognition.ai/introducing-devin.

2024. Factory bringing autonomy to software engineering. https://www.factory.ai/.

2024. Lingma agent. https://github.com/swe-bench/experiments/tree/main/evaluation/lite/20240622_Lingma_Agent.

2024. Marscode code and innovate faster with ai. https://www.marscode.com/.

2024a. Opencsg starship. https://opencsg.com/product?class=StarShip/.

2024b. Opendevin: Code less, make more. https://github.com/OpenDevin/OpenDevin/.

2024. Swe-bench lite. https://www.swebench.com/lite.html.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2024. Repairagent: An autonomous, llm-based agent for program repair. *arXiv preprint arXiv:2403.17134*.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. 2023. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*.

Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, et al. 2024. Coder: Issue resolving with multi-agent and task graphs. *arXiv preprint arXiv:2406.01304*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. 2023. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. In *32nd International Symposium on Software Testing and Analysis (ISSTA)*.

Yinlin Deng, Chunqiu Steven Xia, Chenyuan Yang, Shizhuo Dylan Zhang, Shujing Yang, and Lingming Zhang. 2024. Large language models are edge-case fuzzers: Testing deep learning libraries via fuzzgpt. In *46th International Conference on Software Engineering (ICSE)*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Paul Gauthier. 2024. Aider is ai pair programming in your terminal. `https://aider.chat/`.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with apps. *NeurIPS*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large language models are few-shot testers: Exploring llm-based general bug reproduction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2312–2323. IEEE.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.

Caroline Lemieux, Jeevana Priya Inala, Shuvendu K Lahiri, and Siddhartha Sen. 2023. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models. In *45th International Conference on Software Engineering (ICSE)*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you!

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022a. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022b. Competition-level code generation with alphacode. *Science*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023a. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023b. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*.

Yuliang Liu, Xiangru Tang, Zefan Cai, Junjie Lu, Yichi Zhang, Yanjun Shao, Zexuan Deng, Helan Hu, Zengxian Yang, Kaikai An, et al. 2023c. Mlbench: Large language models leverage open-source libraries for machine learning tasks. *arXiv e-prints*, pages arXiv–2311.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. Starcoder 2 and the stack v2: The next generation. *Preprint*, arXiv:2402.19173.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.

Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. 2024. How to understand whole software repository? *arXiv preprint arXiv:2406.01422*.

Martin Monperrus. 2018. *The living review on automated program repair*. Ph.D. thesis, HAL Archives Ouvertes.

Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sankar, Lambert Pouguem Wassi, Michele Merler, Boris Sobolev, Raju Pavuluri, Saurabh Sinha, and Reyhaneh Jabbarvand. 2024. Lost in translation: A study of bugs introduced by large language models while translating code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Noah Patton, Kia Rahmani, Meghana Missula, Joydeep Biswas, and Işıl Dillig. 2024. Programming-by-demonstration for long-horizon robot tasks. *Proceedings of the ACM on Programming Languages*, 8(POPL):512–545.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. *Advances in neural information processing systems*, 33:20601–20611.

Baptiste Roziere, Jie M Zhang, Francois Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. 2021. Leveraging automated unit tests for unsupervised code translation. *arXiv preprint arXiv:2110.06773*.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024a. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*.

Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. 2024b. Universal fuzzing via large language models. In *46th International Conference on Software Engineering (ICSE)*.

Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the ACM/IEEE 45th International Conference on Software Engineering*, ICSE '23.

Chunqiu Steven Xia and Lingming Zhang. 2023. Keep the conversation going: Fixing 162 out of 337 bugs for $0.42 each using chatgpt. *arXiv preprint arXiv:2304.00385*.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*.

Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Alex Polozov, and Charles Sutton. 2022. Natural language to code generation in interactive data science notebooks.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. Autocoderover: Autonomous program improvement. *Preprint*, arXiv:2404.05427.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. 2023. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2024. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*.

A Appendix

A.1 Categories and Subcategories

Computer Vision and Graphics

- **3D Vision and Reconstruction**
  - Structured3D: Structured 3D modeling dataset for computer vision and graphics (ECCV 2020)
  - RayDF: Light ray-based 3D rendering and learning (NeurIPS 2023)
  - SSDNeRF: Diffusion-based NeRF for 3D reconstruction (ICCV 2023)
  - NeuralPull: Learning distance functions for point clouds (ICML 2021)
  - DAD-3DHeads: Dataset for 3D head alignment and reconstruction (CVPR 2022)
  - LeGO-LOAM: Lidar-based odometry and mapping for robotics
- **Image Segmentation, Processing, and Enhancement**
  - Unsupervised-Semantic-Segmentation: Unsupervised methods for semantic segmentation (ICCV 2021)
  - Seg-Uncertainty: Scene adaptation for self-driving cars (IJCAI 2020, IJCV 2021)
  - RGBD_Semantic_Segmentation_PyTorch: RGBD scene recognition and segmentation (ECCV 2020)
  - BCNet: Occlusion-aware segmentation (CVPR 2021)
  - Deep_Metric: Metric learning for image retrieval
  - Multiview2Novelview: Novel view synthesis from multiple views (ECCV 2018)
  - NeRCo: Low-light image enhancement using neural representations (ICCV 2023)
  - Transfiner: High-quality instance segmentation with mask transformers (CVPR 2022)
- **Pose Estimation and Object Detection**
  - HybrIK: Hybrid kinematics for 3D human pose (CVPR 2021)
  - SmoothNet: Human pose refinement in videos (ECCV 2022)
  - FSA-Net: Head pose estimation from images (CVPR 2019)
  - VisualDet3D: Monocular 3D detection for autonomous driving
  - DEVIANT: Monocular 3D object detection for autonomous driving (ECCV 2022)
  - OWOD: Open World Object Detection with incremental learning (CVPR 2021)

Machine Learning Models and Techniques

- **Generative and Diffusion Models**
  - DDM2: Diffusion models for MRI denoising (ICLR 2023)
  - DDNM: Diffusion models for zero-shot image restoration (ICLR 2023)
  - diffusion-point-cloud: Generating 3D point clouds with diffusion models (CVPR 2021)
  - GeoDiff: Diffusion models for molecular structures (ICLR 2022)
  - probabilistic_unet: Conditional segmentation with probabilistic UNet (NeurIPS 2018)
  - LTSF-Linear: Linear models for time series forecasting (AAAI 2023)
  - unconditional-time-series-diffusion: Diffusion models for time series prediction (NeurIPS 2023)
  - dyffusion: Spatiotemporal forecasting using diffusion models (NeurIPS 2023)
- **Graph Neural Networks and Transformers**
  - GraphSAINT: Scalable graph neural networks with sampling (ICLR 2020, IPDPS 2019)
  - Graph-Transformer: Applying transformers to graph data (WWW 2022)
  - graphtransformer: Generalized transformers for graphs (DLG-AAAI 2021)
  - GPT-GNN: Pre-training techniques for graph neural networks (KDD 2020)
  - GraphMAE: Masked autoencoders for graph learning (KDD 2022)
- **Self-Supervised and Contrastive Learning**
  - skip-connections-matter: Impact of skip connections on adversarial example transferability (ICLR 2020)

- SparK: BERT-style pretraining on convolutional networks (ICLR 2023)
- imbalanced-semi-self: Semi-supervised learning for class-imbalanced datasets (NeurIPS 2020)
- CURL: Image enhancement via neural curve layers (ICPR 2020)

- **Model Interpretability and Optimization**
  - rrl: Rule-based learning for interpretable classification (NeurIPS 2021)
  - gradient-descent-the-ultimate-optimizer: Advanced optimization techniques (NeurIPS 2022)

Large Language Models (LLMs)

- **LLM Development and Optimization**
  - llama-recipes: Fine-tuning scripts for LLaMA models
  - llama3: Official GitHub repository for LLaMA 3
  - alpaca-lora: Instruct-tuning LLaMA on consumer hardware
  - MetaGPT: Multi-agent AI development framework using LLMs
  - Stanford_Alpaca: Training and dataset generation for Alpaca models

- **Inference and Efficiency in LLMs**
  - mistral-inference: Efficient inference for Mistral models
  - PowerInfer: Fast LLM serving on consumer-grade GPUs
  - direct-preference-optimization: Implementation of direct preference optimization (DPO)
  - GPTCache: Efficient caching for LLMs integrated with LangChain and llama_index

- **LLM Applications and Evaluation**
  - alpaca_eval: Automatic evaluation for instruction-following LLMs
  - DART: Enhancing few-shot learning with differentiable prompts (ICLR 2022)
  - LRV-Instruction: Reducing hallucination in multi-modal models (ICLR 2024)
  - alpaca_farm: Simulated framework for reinforcement learning from human feedback (RLHF)
  - FastChat: Open-source platform for training and evaluating large language models

Natural Language Processing (NLP)

- **Text Classification and Information Extraction**
  - HVPNeT: Multimodal entity and relation extraction (NAACL 2022)
  - PURE: Simple approach to relation extraction (NAACL 2021)
  - SimCSE: Contrastive sentence embeddings (EMNLP 2021)
  - Text-GCN: Text classification using graph convolutional networks (AAAI 2019)

- **Multimodal and Knowledge-Based Systems**
  - OntoProtein: Protein function prediction using knowledge graphs (ICLR 2022)
  - tree-of-thought-llm: Structured problem-solving with LLMs (NeurIPS 2023)
  - storm: Automated report generation using large language models
  - SWE-agent: AI-based code bug fixing using LLMs

- **Transfer Learning and Domain Adaptation**
  - SubpopBench: Benchmark for subpopulation shifts in domain generalization (ICML 2023)
  - naacl_transfer_learning_tutorial: Transfer learning techniques in NLP (NAACL 2019)
  - private-transformers: Training transformers with differential privacy
  - imbalanced-regression: Handling data imbalance in regression tasks (ICML 2021)

Domain-Specific Applications

- **Healthcare and Bioinformatics**
  - imbalanced-regression: Addressing imbalances in healthcare data (ICML 2021)
  - hyena-dna: Long-range genomic modeling with Hyena
  - OntoProtein: Protein function prediction with pretraining (ICLR 2022)
  - GeoDiff: Molecular structure generation using diffusion models (ICLR 2022)

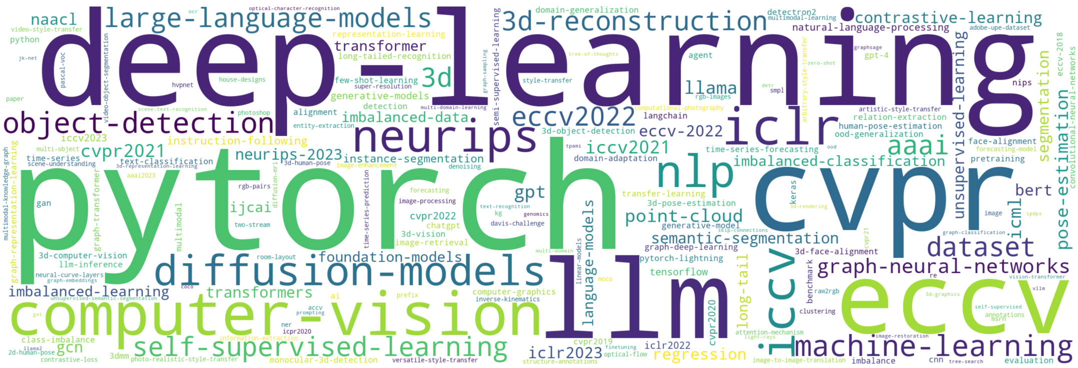

Figure 12: Distribution of Topics in GSRBench100

Table 5: Categories, Subcategories, and Abbreviations

Category	Subcategory	Subcategory Abbreviation
CV	3D Vision & Reconstruction Image Segmentation & Enhancement Pose Estimation & Object Detection	3D Vision & Rec. Img Segm. & Enh. Pose Estim. & Obj Det.
ML	Generative & Diffusion Models Graph Neural Networks & Transformers Self-Supervised & Contrastive Learning Model Interpretability & Optimization	Gen. & Diff. Models GNNs & Transf. Self-Sup. & Contr. Learn. Model Interp. & Opt.
NLP	Text Classification & Extraction Multimodal & Knowledge Systems Transfer Learning & Domain Adaptation	Txt Classif. & Extr. Multi. & Knowl. Sys. Transf. Learn. & Dom. Adap.
LLM	LLM Development & Optimization Inference & Efficiency in LLMs LLM Applications & Evaluation	LLM Dev. & Opt. Infer. & Eff. in LLMs LLM Appl. & Eval.
Interdisciplinary	Healthcare & Bioinformatics Robotics & Autonomous Systems Computational Creativity & Arts	Health. & Bioinf. Robotics & Auto. Sys. Comp. Creativ. & Arts

  - tape: Benchmark tasks for protein sequence modeling

- **Robotics and Autonomous Systems**
  - LeGO-LOAM: Lidar-based odometry and mapping for robotics
  - SuperGlobal: Image retrieval using global features (ICCV 2023)
  - Deep_Metric: Embedding learning for metric-based image retrieval
  - FEARTracker: Robust visual tracking (ECCV 2022)

- **Computational Creativity and Visual Arts**
  - CCPL: Artistic style transfer for images and videos (ECCV 2022)
  - GANgealing: GAN-based visual alignment (CVPR 2022)
  - NeRCo: Low-light image enhancement using neural representations (ICCV 2023)

A.2 Word Cloud of Topics

Figure 12 presents a word cloud that visualizes the frequency of topics across the repositories, highlighting the primary focus areas within GSRBench100. It provides a quick overview of the thematic concentration of the dataset.

B Repository list and Commit ID

We specify the commit ID we used in our benchmark dataset for reproducibility. Please see table 6 and table 7 for details.

C Categories, Subcategories, and Abbreviations

C.0.1 Example Text from README

Below are case studies of how models (GPT-4o, Claude 3, LLaMA 3.1, and Mistral) extract the commands from a Repository.

Table 6: GitHub Repository Details - Part 1

Repository URL	Commit ID	Branch
https://github.com/cleardusk/3DDFA_V2	1b6c67601abffc1e9f248b291708aef0e43b55ae	master
https://github.com/lkeab/BCNet	d6580e8a2a0b5e71c0ae6913ed0340c101d35723	main
https://github.com/JarrentWu1031/CCPL	d0b6b326d7d624b5e8d1543a3a84a745a08fd883	main
https://github.com/sjmoran/CURL	4be9753a8063f9833423e4aa5947ae7a64b114f8	master
https://github.com/tiangexiang/CurveNet	c2e7cf642b7e08d9aec5b70263f6989a85c9e191	main
https://github.com/PinataFarms/DAD-3DHeads	3acc5c2a1177d354a1247c49e44a83ad682ea6a1	main
https://github.com/zjunlp/DART	d418ded8bd548ef25f2d030990e707a497e93483	main
https://github.com/StanfordMIMI/DDM2	d07be20ad36446f8e35621d4b0d92e7cf54c169e	main
https://github.com/wyhuai/DDNM	00b58eac7843a4c99114fd8fa42da7aa2b6808af	main
https://github.com/abhi1kumar/DEVIANT	009955f3bbb21c38a687eaae59bdfcb82eca93e7	main
https://github.com/shenweichen/DSIN	e8ba406eeda0916214897d44866bffc419c3edb0	master
https://github.com/bnu-wangxun/Deep_Metric	04ca51093db13135a04e3c94401bc898c6af0c40	master
https://github.com/tjiiv-cprg/EPro-PnP	42412220b641aef9e8943ceba516b3175631d370	main
https://github.com/PinataFarms/FEARTracker	0a3bd039918909c79c1b7e55a4bfb7807520abde	main
https://github.com/shamangary/FSA-Net	4361d0e48103bb215d15734220c9d17e6812bb48	master
https://github.com/lm-sys/FastChat	92a6d1fcd69a88ea169c0b01065ce44f1e690a2c	main
https://github.com/acbull/GPT-GNN	f26e13c69ddc8a3f2580cb16d0b9a1c73d89f4bc	master
https://github.com/zilliztech/GPTCache	75ab7ec7b871c8399a95d5bf528441f2856250dd	main
https://github.com/MinkaiXu/GeoDiff	ea0ca48045a2f7abfccd7f0df449e45eb6eae638	main
https://github.com/daiquocnguyen/Graph-Transformer	99c88a116148fdaa8d3071fcc548e5c471ae607f	master
https://github.com/THUDM/GraphMAE	b14f080c919257b495e3cb6474286d5252d6a635	main
https://github.com/GraphSAINT/GraphSAINT	c9b1e340d7b951465ac4a9251eef93832e68b003	master
https://github.com/zjunlp/HVPNeT	52c77f7835a295d9c8534997b1316c42b2662972	main
https://github.com/IDEA-Research/HumanSD	c5db29dd66a3e40afa8b4bed630f0aa7ea001880	main
https://github.com/Jeff-sjtu/HybrIK	9b8681dcf3c902dd5dacc01520ba04982990e1e2	main
https://github.com/princeton-nlp/LM-BFF	c282f521001f9c299d29eec7b459266f2b14fbaf	main
https://github.com/FuxiaoLiu/LRV-Instruction	0a5ab538ed96c3e0c9835b5fe02cc8f7fa0bf8fa	main
https://github.com/cure-lab/LTSF-Linear	0c113668a3b88c4c4ee586b8c5ec3e539c4de5a6	main
https://github.com/RobustFieldAutonomyLab/LeGO-LOAM	896a7a95a8bc510b76819d4cc48707e344bad621	master
https://github.com/tfzhou/MATNet	c8b95e527c486c304f711cc7dffb060f31abe19f	master
https://github.com/MIVRC/MSRN-PyTorch	a0e038de7eb42e21d2e88c38e6490b61a02c566e	master
https://github.com/princeton-nlp/MeZO	552cb1b710767f9a6e1dc8f9645d7640376f9941	main
https://github.com/geekan/MetaGPT	5446c7e490e7203c61b2ff31181551b2c0f4a86b	main
https://github.com/shaohua0116/Multiview2Novelview	a5e236f3c564bf287c8a09d855fd2134ba86b299	master
https://github.com/Ysz2022/NeRCo	6b0e1112231d0902976ad76357044de582a307f3	main
https://github.com/mabaorui/NeuralPull	c093a52308a9b74446d24cc6c1b0fee5ee5bb7bb	master
https://github.com/JosephKJ/OWOD	23890f188cd1a6801c6ac0e3dacd78b8572b8c29	master
https://github.com/zjunlp/OntoProtein	6360f458e11670ecfaf853ee68f2087b31439dc0	main
https://github.com/princeton-nlp/PURE	b1e9cad39bec10eb3c355dc5a8e4e75dd0afebf5	main
https://github.com/SJTU-IPADS/PowerInfer	61cac9bf25e60336bbad27ada9dbb809204473ac	main
https://github.com/charlesCXK/RGBD_Semantic_Segmentation_PyTorch	32b3f86822d278103a13ea6f93f9668d3b631398	master
https://github.com/vLAR-group/RayDF	ca6c663523b777732788a5d8100d36251a482b31	master
https://github.com/Paranioar/SGRAF	50d0c6f9caf759099b28371046f780342357c405	main
https://github.com/Lakonik/SSDNeRF	b9d195db76bb715c475b24287362d9627d77d3bb	main
https://github.com/shamangary/SSR-Net	f98b6cbe1c9c8c78649e5a331f94113564521525	master
https://github.com/princeton-nlp/SWE-agent	36e430d27ffd11269738df92d6c521cab2207dcb	main
https://github.com/daveredrum/ScanRefer	9d7483053e8d29acfd4db4eb1bc28f1564f5dddb	master
https://github.com/layumi/Seg-Uncertainty	6fce9eae141c2c0592b3e7c1b3e5f8ee7b1ce9a6	master
https://github.com/princeton-nlp/SimCSE	7edb07e05cec0d5293fc1696b578d8056dba76ef	main
https://github.com/cure-lab/SmoothNet	c03e93e8a14f55b9aa087dced2751a7a5e2d50b0	main

Table 7: GitHub Repository Details - Part 2

Repository URL	Commit ID	Branch
https://github.com/keyu-tian/SparK	a63e386f8e5186bc07ad7fce86e06b08f48a61ea	main
https://github.com/bertjiazheng/Structured3D	d06d1b241b290ae7ed6b311d710ffafffcb567de	master
https://github.com/YyzHarry/SubpopBench	4d3dbbe21029666ef19d040e110ec22908640c5b	main
https://github.com/ShihaoShao-GH/SuperGlobal	86946964b907e6f28a7264add2c15640fae30009	main
https://github.com/codeKgu/Text-GCN	70b970a52efcb80235cf0ae3e578eaf80278d5f3	master
https://github.com/wvangansbeke/Unsupervised-Semantic-Segmentation	dfd5fa0a1542f2b26824b4059bd2bb1240c5c94b	main
https://github.com/tloen/alpaca-lora	8bb8579e403dc78e37fe81ffbb253c413007323f	main
https://github.com/tatsu-lab/alpaca_eval	32c8c0d068205c38b02003b67e0beec82a8f1ac2	main
https://github.com/tatsu-lab/alpaca_farm	30717ddae735365de756ee2085191b491a71788d	main
https://github.com/adobe/antialiased-cnns	b27a34a26f3ab039113d44d83c54d0428598ac9c	master
https://github.com/richardaecn/class-balanced-loss	1d7857208a2abc03d84e35a9d5383af8225d4b4d	master
https://github.com/luost26/diffusion-point-cloud	1e30d48d018820fbc7c67c8b3190215bd41878e4	main
https://github.com/eric-mitchell/direct-preference-optimization	f8b8c0f49dc92a430bae41585f9d467d3618fe2f	main
https://github.com/Rose-STL-Lab/dyffusion	832574f6f788a0cd4a4d75e8f59b3c07c7e8446b	main
https://github.com/wpeebles/gangealing	ffa6387c7ffd3f7de76bdc693dc2272e274e9bfd	main
https://github.com/openai/gpt-3	d7a9bb505df6f630f9bab3b30c889e52f22eb9ea	master
https://github.com/kach/gradient-descent-the-ultimate-optimizer	b3b047e02ca6d32e0e61e34a0ca6e0bc57e55bdf	main
https://github.com/microsoft/graphrag	61b5eea34783c58074b3c53f1689ad8a5ba6b6ee	main
https://github.com/graphdeeplearning/graphtransformer	c9cd49368eed4507f9ae92a137d90a7a9d7efc3a	main
https://github.com/HazyResearch/hyena-dna	d553021b483b82980aa4b868b37ec2d4332e198a	main
https://github.com/YyzHarry/imbalanced-regression	a6fdc45d45c04e6f5c40f43925bc66e580911084	main
https://github.com/YyzHarry/imbalanced-semi-self	b91ad29fd8805ddf0a146f735905b0c869e68ae4	master
https://github.com/meta-llama/llama	8fac8befd776bc03242fe7bc2236cdb41b6c609c	main
https://github.com/meta-llama/llama-recipes	8c1418e93b817cb6734a9cfe095b270f5a0f48f5	main
https://github.com/meta-llama/llama3	18f515a3c3c5f02cf45c6ac56cc5d039488e867a	main
https://github.com/state-spaces/mamba	a71bb5a83bfa289b5807aefc1767232dee77b35e	main
https://github.com/kwotsin/mimicry	a7fda06c4aff1e6af8dc4c4a35ed6636e434c766	master
https://github.com/mistralai/mistral-finetune	5b8adb54a1263664d52dab6f94581bf24d7b59e3	main
https://github.com/mistralai/mistral-inference	1f583071dc7aad2ca35cb9896140316ffece5b65	main
https://github.com/YyzHarry/multi-domain-imbalance	2efbfefd34542e365293f798d79f70cee5e54303	main
https://github.com/JiangWenPL/multiperson	e8ae029cc691f3f9c3958a23f762f3d72cf65c54	master
https://github.com/zju3dv/mvpose	38b958f423f2de2bf7562f5a386c27440eab8c53	master
https://github.com/huggingface/naacl_transfer_learning_tutorial	dc976775bb11edee24a77e2ce161450089c5e169	master
https://github.com/baudm/parseq	1902db043c029a7e03a3818c616c06600af574be	main
https://github.com/lxuechen/private-transformers	18ccc4eab7355e4ac96051a82434796f6aa4624b	main
https://github.com/SimonKohl/probabilistic_unet	7a2e79d549184d0f3a47d0deaa054a70b0f54a3f	master
https://github.com/Jeff-sjtu/res-loglikelihood-regression	203dc3195ee5a11ed6f47c066ffdb83247511359	master
https://github.com/12wang3/rrl	f8d0886b23c4e15f63c62c248b97d4eb73386ad1	main
https://github.com/csdongxian/skip-connections-matter	9b2e5cca9b673efcac253e16b2f55f6cda1a8692	master
https://github.com/xuchen-ethz/snarf	ae0c893cc049f0f8270eaa401e138dff5d4637b9	main
https://github.com/tatsu-lab/stanford_alpaca	761dc5bfbdeeffa89b8bff5d038781a4055f796a	main
https://github.com/stanford-oval/storm	263d894e34e2a399a18f80fbf3c9028dc06e1a1d	main
https://github.com/songlab-cal/tape	6d345c2b2bbf52cd32cf179325c222afd92aec7e	master
https://github.com/SysCV/transfiner	5b61fb53d8df5484f44c8b7d8415f398fd283ddc	main
https://github.com/yzhq97/transmomo.pytorch	0e4d2f0e7e8af159e018c8a85f9d789de963a83a	master
https://github.com/princeton-nlp/tree-of-thought-llm	ab400345c5ea39d28ea6d7d3be0e417b11113c87	master
https://github.com/amazon-science/unconditional-time-series-diffusion	3eafeffdffefd4d2f5344ac63e5627a746c331a3	main
https://github.com/elliottwu/unsup3d	dc961410d61684561f19525c2f7e9ee6f4dacb91	master
https://github.com/dddzg/up-detr	44c0c6eb4bddf409a41cbaae31b7360062495199	master
https://github.com/Owen-Liuyuxuan/visualDet3D	ad229f2e491cba2c48f9bb2c211fe672294e2398	master

REPOSITORY ID: 3DDFA_V2

README

```
Requirements
See requirements.txt, tested on macOS and Linux platforms. The Windows users
may refer to FQA for building issues. Note that this repo uses Python3. The
major dependencies are PyTorch, numpy, opencv-python, and onnxruntime, etc.
If you run the demos with --onnx flag to do acceleration, you may need to
install libomp first, i.e., brew install libomp on macOS.

Usage

# 1. Clone this repo
git clone https://github.com/cleardusk/3DDFA_V2.git
cd 3DDFA_V2

# 2. Build the cython version
sh ./build.sh

# 3. Run demos
# Running on a still image
python3 demo.py -f examples/inputs/emma.jpg --onnx

# Running on videos
python3 demo_video.py -f examples/inputs/videos/214.avi --onnx

# Running on webcam
python3 demo_webcam_smooth.py --onnx

The implementation of tracking is simply by alignment. If the head pose > 90
degrees or the motion is too fast, the alignment may fail. A threshold is
used to trickly check the tracking state, but it is unstable.
```

GPT-4o:
Extracted commands for Environment Setup / Requirement / Installation:

```
git clone https://github.com/cleardusk/3DDFA_V2.git; cd 3DDFA_V2; sh ./build.sh
```

Error message from bash shell:

```
"stderr": "Traceback (most recent call last):\n  File \"/workspace/3DDFA_V2/FaceBoxes/
utils/build.py\", line 15, in <module>\n    from Cython.Distutils import build_ext\n
ModuleNotFoundError: No module named 'Cython'\nTraceback (most recent call last):\n
File \"/workspace/3DDFA_V2/Sim3DR/setup.py\", line 7, in <module>\n    from Cython.
Build import cythonize\nModuleNotFoundError: No module named 'Cython'",

Analyzer suggested solution:
"command": "git clone https://github.com/cleardusk/3DDFA_V2.git; cd 3DDFA_V2;
pip install cython; sh ./build.sh"
```

Table 8: Commands and errors for setting up 3DDFA_V2 repository using GPT-4o.

LLaMA 3.1:
Extracted commands for Environment Setup / Requirement / Installation:

```
pip install -r requirements.txt
brew install libomp
```

Claude 3:
Extracted commands for Environment Setup / Requirement / Installation:

```
git clone https://github.com/cleardusk/3DDFA_V2.git
cd 3DDFA_V2
sh ./build.sh
```

Error message from bash shell:

```
"stderr": "Traceback (most recent call last):\n  File \"/workspace/3DDFA_V2/FaceBoxes/
utils/build.py\", line 15, in <module>\n    from Cython.Distutils import build_ext\n
ModuleNotFoundError: No module named 'Cython'\nTraceback (most recent call last):\n
File \"/workspace/3DDFA_V2/Sim3DR/setup.py\", line 7, in <module>\n    from Cython.
Build import cythonize\nModuleNotFoundError: No module named 'Cython'",

Analyzer suggested solution:
"command": "pip install Cython && sh ./build.sh",
```

Mistral:
Extracted commands for Environment Setup / Requirement / Installation:

```
git clone https://github.com/cleardusk/3DDFA_V2.git && cd 3DDFA_V2 && sh ./build.sh &&
brew install libomp
```

Error message from bash shell:

```
"stderr": "bash: brew: command not found"

Analyzer suggested solution:
"command": "sudo apt-get install homebrew && git clone https://github.com/cleardusk/
3DDFA_V2.git && cd 3DDFA_V2 && sh ./build.sh && brew install libomp",
```

Table 9: Commands and errors for setting up 3DDFA_V2 repository using LLaMA 3.1, Claude 3 and Mistral.